Memory repair architectures for high defect densities

Panagiota Papavramidou

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Nano-Électronique et Nano-Technologies**

Arrêté ministériel : 7 août 2006

Présentée par

Panagiota PAPAVRAMIDOU

Thèse dirigée par **Michael NICOLAIDIS**

préparée au sein du **Laboratoire TIMA, Équipe ARIS**
dans **l'École Doctorale Électronique, Électrotechnique, Automatique et Traitement du Signal**

Architectures de réparation des mémoires pour des hautes densités des défauts

Thèse soutenue publiquement le **19 Novembre 2014,**
devant le jury composé de :

Michel RENOVELL,
Directeur de Recherche CNRS, LIRMM, Président
Paolo, PRINETTO
Professeur, Politecnico di Torino, Rapporteur
Yervant, ZORIAN
Chief Architect, Synopsys US, Rapporteur
Emmanuel, SIMEU
Maître de conférences, Invité
Michel, NICOLAIDIS
Directeur de Recherche CNRS, TIMA, Directeur de thèse

Στο Λευτερη, το Μιχαλη, και τους Γονεις μου

Resumé

I Introduction, Stratégie, et Motivation

A cause d'une miniaturisation extrême, les technologies CMOS ultimes et post-CMOS seront impactées par des densités des fautes de fabrication et de vieillissement très élevées, et seront en même temps affectées par des densités de puissance dissipé et par de températures très élevées, nécessitant ainsi des techniques de réparation, de tolérance aux fautes en ligne, et de faible consommation, très poussées. Dans ce contexte, le développement d'une solution efficace pour l'autoréparation des grandes densités des défauts devient d'importance capitale car elle permettra :

i. Réparer les grands nombres des fautes induites par la fabrication, et améliorer ainsi le rendement de fabrication.
ii. Prolonger la vie du circuit en réparant les grands nombres de fautes induites par le vieillissement au fur et à mesure de leur apparition.
iii. Disposant une technique d'autoréparation pour des densités des défauts bien plus élevées que celles requises pour les fautes de fabrication et les fautes du vieillissement, permettra aussi de réduire de façon drastique la puissance dissipée. En effet, dans ce cas nous allons pouvoir réduire de façon agressive le niveau de la tension d'alimentation (Vdd) et réparer les cellules qui deviendront défaillantes à cause de cette réduction. Grace à cette réparation, la mémoire fonctionnera correctement à ce niveau de tension très bas, qui pourra être utilisé pendant l'exécution de l'application afin d'obtenir une réduction drastique de la consommation (i.e. proportionnelle au carrée de la réduction du Vdd).
iv. Restent finalement les problèmes de fiabilité induites par les fautes du vieillissement survenant pendant l'exécution d'une application, lesquelles, à partir de leur apparition et jusqu'à la prochaine séance de test/réparation, affecteront l'exécution correcte de l'application. Disposant une technique d'autoréparation pour des densités des défauts encore plus élevées que celles requises pour les trois classes des fautes prises en compte ci-dessous (points i, ii, et iii), permettra de résoudre ce problème de fiabilité en testant la mémoire sous des conditions bien plus contraignantes que les pires conditions (niveau de la tension d'alimentation et vitesse de fonctionnement) pouvant être utilisées pendant l'exécution des applications. En effet, ce test permettra de détecter et réparer préventivement (i.e. avant qu'elles commencent à induire des erreurs dans l'application) les cellules qui ne sont pas encore défaillantes mais pourraient potentiellement le devenir avant la prochaine séance de test et réparation.

Les techniques d'autoréparation des mémoires sont utilisées actuellement pour l'amélioration du rendement de fabrication, et accessoirement de la durée de vie. Ainsi, un premier apport de cette thèse consiste à l'extension décrite ci-dessus des domaines d'application de réparation des mémoires, de façon à résoudre un ensemble de problèmes plus vaste que d'habitude (i.e. pas seulement l'amélioration du rendement de fabrication et l'augmentation de la durée de vie, mais aussi la réduction de la puissance dissipée et l'amélioration de la fiabilité). *Cette stratégie originale ouvre la possibilité de régler l'ensemble des problèmes induits par la miniaturisation nanométrique par le seul biais de la réparation des mémoires pour des grandes densités des défauts, et procure de cette façon une importance capitale aux efforts de développement des techniques efficaces permettant ce type de réparation*. Ceci et la motivation fondamentale des travaux présentés dans cette thèse.

La seule technique connue permettant une réparation à faible coût des mémoires affectées par des grandes densités des défauts, combine l'utilisation des codes ECC avec la réparation des mots contenant des fautes multiples (« ECC-based memory repair » en anglais). Un deuxième apport de cette thèse consiste à avoir montré que la technique de «ECC-based memory repair» perdait son intérêt à cause du coût très élevé de la circuiterie de diagnostic des fautes. Nous mettons aussi en évidence un autre problème de l'autoréparation qui se manifeste dans le cas des grandes densités des fautes. En effet, dans une mémoire utilisant l'autoréparation, à chaque accès mémoire l'adresse courante est comparée avec toutes les adresses nécessitant réparation. Pour des très grandes densités des défauts le nombre des cites défaillants devient très grand, induisant un nombre de comparaisons très grand, et par conséquent une consommation importante même dans le cas du «ECC-based memory repair» qui réduit de façon drastique le nombre des cites nécessitant réparation.

Suite au constat de ces problèmes, mis en évidence par une analyse préliminaire présentée dans le Chapitre 1 de cette thèse, les contributions de la thèse, présentées dans les Chapitres 2, 3, 4, 5, et 6, consistent au développement d'une panoplie des solutions permettant leur résolution.

II Contributions

II.1 Algorithmes de Test des Mémoires pour l'approche de ECC-Based Repair

Le premier problème que nous avons traité concerne le coût élevé induit par la circuiterie de diagnostic, nécessaire dans le cadre du ECC-based repair. Ce problème est du au fait que, l'approche du ECC-based repair nécessite la localisation des mots mémoire contenant plus d'une cellule défaillante. Néanmoins, les algorithmes existants de test des mémoires garantissent la détection de tous les fautes du modèle cible, mais en cas de fautes multiples affectant le même mot mémoire, la détection de ces fautes peut se faire dans des séquencés différentes de l'algorithme de test. Ainsi, au moment de chaque détection d'une faute, il n'est pas certain qu'il n'y a pas d'autres fautes dans le même mot. Il devient ainsi nécessaire de mémoriser dans une CAM les adresses de tous les mots défaillants et les positions de cellules défaillantes, afin de pouvoir par la suite déterminer quels mots contiennent des fautes multiples. Mémoriser tous les mots défaillants nécessite une CAM de diagnostic très large. Ainsi, la taille élevée de la *CAM de*

diagnostic élimine les gains en surface obtenus grâce à la forte réduction de la taille de la *CAM de réparation* obtenue grâce au ECC-based repair.

Afin d'éliminer la *CAM du diagnostic*, dans le Chapitre 2 nous proposons une nouvelle famille d'algorithmes de test que nous avons nommé SRDF (pour Single-Read Double-Fault detection), qui garantissent la détection dans la même opération de lecture d'au moins deux des fautes affectant le même mot mémoire. Ainsi il n'est plus nécessaire de mémoriser tous les mots défaillantes afin d'identifier les mots comportant plusieurs cellules défaillantes, car chacun de ces mots est reconnu immédiatement lors d'une opération de lecture. Grace à cette propriété, la CAM de diagnostic est complètement éliminée, permettant la réduction drastique du coût en surface de ECC-based memory repair. Néanmoins, cette propriété augmente exponentiellement le nombre des états défaillants qui doivent être pris en compte par l'algorithme de test de la mémoire, et rend extrêmement complexe le développement de ce type d'algorithmes. Malgré ces difficultés de taille, nous avons réussie notre pari en développant une approche théorique (de loin la plus complexe dans toute l'histoire des algorithmes de test pour mémoires), permettant de proposer des algorithmes de test vérifiant cette propriété et prouver leur validité.

Pour illustrer notre approche, nous avons considéré le model étendu des fautes de mémoire, comprenant les fautes qui ne mettent pas en jeux des interactions entre cellules (single–cell FFMs), ainsi que les fautes mettant en jeux des interactions entre deux cellules (double–cell FFMs), présentées dans les tableaux I, et II suivant leur classification systématique proposée en [29]. Pour des raisons de compacité, dans ces tableaux nous avons remplacé : l'état de la cellule «agresseur» (qu'il soit 0 ou 1) par *a*; et l'état de la cellule «victime» (qu'il soit 0 ou 1) par *v*; et le symbole de transition de la cellule «victime» (qu'il soit ↑ or ↓) par ◊.

Tableau I. Liste de single-cell FFMs

#	FFM	Fault Primitives
1	SF	$< v/\bar{v}/->$
2	TF	$< \bar{v}\,wv/\bar{v}/->$
3	WDF	$<vwv/◊/->$
4	RDF	$<rv/◊/\bar{v}>$
5	DRDF	<rv/◊/v>
6	IRF	$rv/v/\bar{v}>$

Tableau II. Liste de two-cell FFMs

#	FFM	Fault Primitives
1	CFst	$<a; v/\bar{v}/->$
2	CFds:	
2.1	CFds(r*a*)	$<ra; v/◊/->$
2.2	CFds($a\,w\,\bar{a}$)	$<a\,w\,\bar{a}; v/◊/->$
2.3	CFds(*a*w*a*)	$<awa; v/◊/->$
3	CFtr	$<a; vw\bar{v}/v/ - >$
4	CFwd	$<a; vwv /◊/ - >$
5	CFrd	$<a; rv/◊/\bar{v}>$
6	CFdrd	$<a; rv/◊/v >$
7	CFir	$< a; rv /v/\bar{v}>$

Afin d'assurer la propriété SRDF nous avons développé plusieurs algorithmes de test. Pour ces développements nous avons considéré des algorithmes de type March, car ils sont moins complexes en termes de longueur du test, et aussi très réguliers et permettent ainsi une implémentation BIST (Built-In Self-Test) peu couteuse.

Dans un premier temps nous avons développé l'algorithme March SRDF1, présenté dans la figure I, et nous avons démontré qu'en exécutant cet algorithme pour chaque vecteur V_i d'un ensemble de vecteurs

"2-covering"[1], assure cette propriété pour chaque faute double [f1, f2] telle que f1 et f2 appartiennent à l'ensemble des fautes comprenant les single-cell FFMs et les two-cell FFMs des types CFst and CFds.

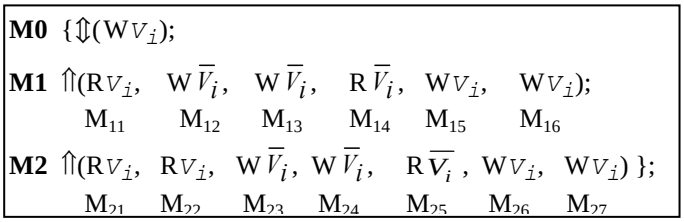


Figure I. March SRDF1

Dans le cas de nombreuses applications cette couverture de fautes sera insuffisante. Ainsi nous avons aussi développé les algorithmes March SRDF3, March SRDF4, et March SRDF5 (montés dans le figures II, III, et IV), et nous avons démontré que l'exécution de March SRDF3 pour un ensemble de vecteurs "2-covering", March SRDF4 pour un ensemble de vecteurs "4-covering"[2], et March SRDF5 pour un ensemble de vecteurs "3-covering"[3], assure la propriété SRDF pour :

- 100% des fautes de multiplicité plus grande que 2.
- 99,97% des fautes de multiplicité égale à 2.

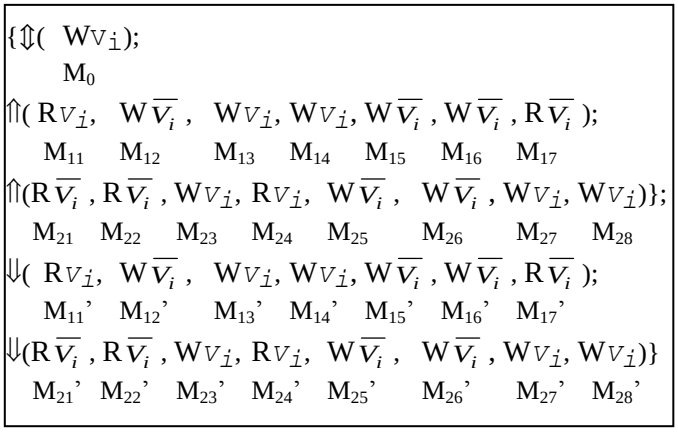


Figure B. March SRDF3

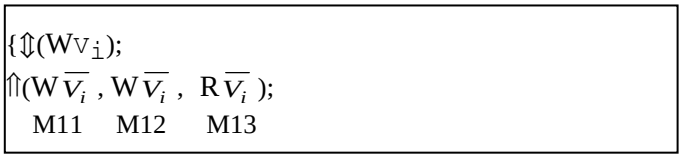


Figure C: March SRDF4

[1] Un ensemble de vecteurs binaires est appelé "2-covering" s'il applique sur chaque pair des positions tous les combinaisons des valeurs binaires (c'est à dire les combinaisons 00, 01, 10, et 11).

[2] Un ensemble de vecteurs binaires est appelé "4-covering" s'il applique sur chaque combinaison de 4 positions toutes les 16 combinaisons des valeurs binaires de 4 bits.

[3] Un ensemble de vecteurs binaires est appelé "3-covering" s'il applique sur chaque combinaison de 3 positions toutes les 8 combinaisons des valeurs binaires de 3 bits.

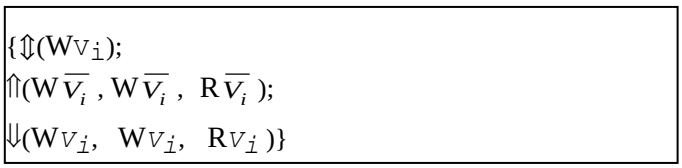


Figure IV: March SRDF5

Cette couverture de fautes est très élevée et devrait satisfaire la grande majorité d'applications. Néanmoins, si une application exige une couverture de fautes plus élevée, nous avons proposé l'approche suivante afin de protéger l'application pour le 0,03% des fautes doubles non-couvertes.

Cette approche exploite le fait que les fautes non-couvertes sont doubles. Par consequent, ils produisent des erreurs qui sont toujours détectables par le code ECC. *Ainsi nous pouvons utiliser la stratégie suivante* : Si une des ces fautes affecte un mot de la mémoire et ce mot n'est pas identifiée de contenir deux fautes par les algorithmes de test il reste non-réparée. Dans la suite si cette faute produite une erreur double pendant l'exécution de l'application, elle sera détectée par le code ECC et sera réparée. Néanmoins, cette stratégie pose le problème de fiabilité suivant. Si un "soft-error" affecte le mot contenant le double faute avant que ce mot aurait produit une erreur double et soit détectée et réparée, une erreur triple pourrait se produire. Comme cette erreur dépassé la capacité de détection du code ECC elle pourra induire une défaillance dans l'application. Néanmoins, pour un produit qui contient une très large mémoire de capacité de 100 Gbit et qui est affecté par une très grande densité de fautes de l'ordre de 10^{-3}, nous avons montré que la probabilité d'occurrence de ce type d'évènement pour l'ensemble de la durée de vie de ce produit est égale à $7.1X10^{-8}$, et ceci est valable aussi grande qu'elle soit la durée de sa vie. Ainsi, sur 10 millions de ces produits, moins d'un produira une défaillance dans l'application pendant la totalité de la durée de sa vie. Ce qui reste extrêmement faible.

Les résultats d'évaluation de l'approche utimisant les algorithmes SRDF sont présentés dans les tableaux III et IV.

Le tableau III compare les coûts en surface et en consommation de la technique de réparation conventionnelle avec les coûts de la technique du ECC-based repair utilisant des algorithmes de test de type SRDF. Dans ce tableau, nous considérons un SOC ayant une capacité totale de mémoire de 9,75 Gbit. La colonne 1 du tableau donne la densité des défauts considérée (correspondant à la probabilité d'une cellule mémoire d'être défaillante), et la colonne 2 donne le nombre des SRAM embarquées sur lesquelles est distribuée la capacité totale de mémoire de 9,75 Gbit. Les colonnes 3, 4, et 5 donnent les résultats pour la réparation conventionnelle : la colonne 3 donne le nombre des mots de CAM nécessaires pour obtenir le succès de réparation visé (i.e. 90%), et les colonnes 4 and 5 donnent les coûts en surface et en consommation de puissance. Les colonnes 6, 7, tet 8 donnent les résultats pour l'approche de ECC-based repair employant des algorithmes de test SRDF: la colonne 6 présente le nombre des mots CAM nécessaires pour obtenir le succès de réparation visé (i.e. 90%), et les colonnes 7 et 8 donnent les coûts en surface et en consommation de puissance.
Ces résultats montrent que l'approche proposée permet une réduction drastique coûts en surface et en consommation de puissance.

Tableau III. Comparaison des coûts en surface et en puissance

P_f	Embed. Mem	Conventional Repair			ECC-based Repair		
		N_{CW}	%A	%P	N_{CW}	%A	%P
10^{-4}	300	3466	1.32	185.3	16	0.008	1.267
	3000	402	1.27	67.90	6	0.028	1.297
3x10 -4	300	10285	3.93	532.9	83	0.036	5.337
	3000	1121	3.46	177.9	17	0.078	3.676
10^{-3}	300	35325	12.75	1629	720	0.249	39.56
	3000	3693	13.49	581.5	98	0.344	17.56

Concernant la durée du test, pour une mémoire ayant des mots de 39-bits (32 bits des donnés et 7 bits de codage ECC), nous trouvons que la longueur des algorithmes SRDF (en nombre d'opérations de lecture et d'écriture) est égale à 873 Nw, où Nw est le nombre des mots de la mémoire. Cette longueur est 46 fois plus large que la durée de l'algorithme de test conventionnel (i.e. non-SRDF) proposé à [31], qui est un algorithme optimale pour les fautes données dans les tableaux I et II. Néanmoins, une comparaison pertinente doit considérer la durée du test plutôt que sa longueur en nombre d'opérations. Cette durée dépend aussi de la consommation des mémoires sous test.

Comme la consommation de puissance pendant une session de test et de réparation est plus grande pour l'approche de réparation conventionnelle, sa durée de test est augmentée proportionnellement, car la puissance maximale permissible dans le SoC va permettre de tester simultanément un nombre de mémoires proportionnellement moindre. Ainsi, en utilisant les résultats de la consommation de puissance présentés dans le tableau III, nous trouvons pour les algorithmes de test SRDF l'augmentation du temps de test effectif présentée dans le tableau IV.

Dans ce tableau, la colonne 1 présente la densité des défauts, la colonne 2 présente le nombre des SRAM embarquées sur lesquelles est repartie la capacité de mémoire totale de 9,75 Gbit; la colonne 3 donne la consommation de puissance totale de ces mémoires embarquées pour l'approche de réparation conventionnelle (i.e. la consommation de la SRAM sous réparation plus la consommation de la CAM de réparation utilisée dans l'approche de réparation conventionnelle) divisée par la puissance de la SRAM; la colonne 4 donne la consommation de puissance totale pour l'approche de ECC-based repair utilisant les algorithmes de test SRDF (i.e . la consommation de la SRAM sous réparation plus la consommation de la CAM de réparation utilisée dans l'approche de ECC-based repair) divisée par la consommation de la SRAM; et la colonne 5 donne l'augmentation de la durée de test correspondante à l'approche de ECC-based repair utilisant les algorithmes de test SRDF. Cette augmentation est déterminée par 46x(consommation de puissance totale dans l'approche de ECC-based repair)/(consommation de puissance totale dans l'approche de réparation conventionnelle).

Tableau IV. Augmentation de la durée du test

P_f	#Embedded Memories	Conventional Repair	ECC-based Repair	
		Total Power	Total Power	Test-time increase
10^{-4}	300	2.85	1.013	16.35
	3000	1.68	1.013	27.7
$3x10^{-4}$	300	6.33	1.053	7.65
	3000	2.78	1.037	17.15
10^{-3}	300	17.29	1.395	3.71
	3000	6.81	1.175	7.94

Nous observons que même dans le scenario de pire cas (i.e. pour Pf = 10^{-4} et pour 3000 mémoires embarquées), dans lequel l'augmentation de la durée de test dans le tableau IV prend sa plus grande valeur (27.7) et l'augmentation de la consommation de puissance pour l'approche de réparation conventionnelle prend sa plus petite valeur (67.9% dans le tableau III), l'approche proposée est clairement plus attractive étant donné que cette augmentation de puissance est complètement inacceptable. De plus, l'avantage de l'approche proposée devient décisif pour des densités de défauts plus élevées. Ainsi, pour Pf = 10^{-3} et 300 mémoires embarquées, la durée de test est augmentée d'un facteur 3.71, qui est nettement préférable que l'énorme augmentation de la consommation de puissance induite par l'approche de réparation conventionnelle, qui est dans ce cas égale à 1629%. Aussi, l'augmentation en surface de 12.75% induite dans ce cas par l'approche conventionnelle est très pénalisante. En effet, comme les mémoires occupent la plus grande partie des SoCs modernes (plus que 90% de la surface du SoC dans la plupart des cas), l'augmentation en surface de 12.75% induite par l'approche conventionnelle représente plus que 11.5% de la surface totale du SoC.

II.2 Algorithm de Diagnostic Itérative pour ECC-based Memory Repair

Les algorithmes SRDF, présentés dans la section précédente, éliminent complètement la circuiterie du diagnostic et réduisent ainsi drastiquement le coût en surface de l'approche de ECC-based repair. Néanmoins, ces algorithmes augmentent la durée de test, car ils sont plus complexes que les algorithmes conventionnels de test des mémoires. Ainsi, nous avons développé une approche alternative, décrite dans le Chapitre 3, qui consiste à une technique originale de diagnostic itérative. Au lieu d'éliminer complètement la CAM de diagnostic, comme le fait l'approche décrite dans la section précédente, cette technique réduit la taille de cette CAM de la façon suivante : l'algorithme de test conventionnel est exécuté itérativement plusieurs fois ; à chaque itération on diagnostique un sous-ensemble des mots mémoire défaillants (ceux qu'ils sont stockées dans la CAM jusqu'au moment où elle est complètement remplie) ; à la fin de l'itération courante on efface certains mots défaillants stockés dan la CAM afin de libérer de l'espace CAM et pouvoir traiter des nouveaux mots défaillants lors de l'itération suivante.

Le premier défi de cette approche est le suivant : quand un mot défaillant est éliminée de la CAM lors d'une itération, et il est détecté à nouveau lors d'une prochaine itération, des informations perdues lors de son effacement peuvent induire un diagnostic erroné. Pour résoudre ce problème nous avons développé un *algorithme de diagnostic qui finalise le diagnostic de l'ensemble des mots défaillants mémorisés dans la CAM lors de chaque itération*, avant les effacer. Ainsi, *nous effaçons des mots défaillants pour lesquels nous avons vérifié de façon certaine qu'elles comportent une seule cellule défaillante*.

Le deuxième défi de cette approche est le suivant : comment ignorer lors d'une itération les mots défaillants effacés lors des itérations précédentes, étant donné que nous ne les connaissons pas (car nous ne possédons plus des informations les concernant). Pour résoudre ce problème nous avons développé *un algorithme de diagnostic, qui commence à mémoriser des mots défaillants dans la CAM à partir du cycle du test dans lequel nous avons stoppé d'ajouter dans la CAM des nouveaux mots défaillants lors de l'itération précédente*.

Finalement, *afin de réduire la durée du test, nous n'exécutons pas à chaque nouvelle itération l'algorithme de test depuis son début*. Mais *cette approche risque de masquer certaines fautes qui seraient sensibilisées par des opérations appartenant à la partie non exécutée de l'algorithme de test*. Pour résoudre ce problème, nous montrons que la sensibilisation des fautes détectés lors d'une séquence d'un algorithme

de test peut être réalisée uniquement pendant cette séquence elle même ou pendant la séquence qui la précède. Ainsi, *à chaque nouvelle itération nous commençons l'exécution de l'algorithme de test à partir de la séquence qui précède la séquence dans laquelle nous commençons à mémoriser des mots défaillants dans la CAM*. De cette façon nous réduisons la durée de test sans masquer des fautes.

Un autre défi majeur concerne l'évaluation de cette approche. Cette évaluation nécessite d'effectuer un grand nombre d'injections de fautes, afin d'obtenir des résultats statistiquement significatifs, et simuler, pour chacune de ces injections, la mémoire défaillante et le circuit de diagnostic en exécutant l'algorithme du test et du diagnostic. Comme la simulation des fautes est un processus très gourmant en temps de calcul (et c'est aussi le cas pour la simulation algorithmique des CAM), *nous avons développé une approche originale que nous appelons « pseudo-simulation de fautes »,* qui réduit considérablement le temps de simulation tout en fournissant des résultats identiques à la simulation des fautes classique. Dans cette approche, *au lieu d'injecter des fautes dans la mémoire, nous injectons ce que nous avons appelé des profiles de détection*. Cette approche consiste à :

- Identifier les emplacements de détection de chaque faute dans l'algorithme de test (profile de détection de la faute).
- Injecter de façon probabiliste ces profiles de détection dans les cellules de la mémoire, et créer un tableau qui contient le résultat de cette injection.
- Développer un algorithme de «pseudo-simulation» qui simule le processus du diagnostique en utilisant ce tableau. Afin d'éviter l'effort de développement de l'algorithme de «pseudo-simulation» pour chaque algorithme de test des mémoires, *nous avons développé un algorithme générique qui prend en entrée l'algorithme de test de la mémoire et génère l'algorithme de «pseudo-simulation» dédié à cet algorithme de test*.

Comme l'approche de pseudo-simulation réduit le temps de simulation de façon drastique, nous avons pu effectuer des campagnes intensives d'injection et simulation des fautes statistiques, permettant d'obtenir des résultats statistiquement significatifs. En effet, pour chaque cas d'étude (capacité de SRAM, de taille de CAM, et de densité des défauts) nous avons effectué 1000 campagnes d'injection des fautes et de simulation. Les résultats sont présentés dans le tableau V.

Tableau V. Coûts en surface, en consommation, et en longueur de test

P_f	#Emb Mem	**Appr. 1** Non-ECC Repair		All ECC Repair	**Appr. 2** ECC-Rep Separate CAM	**Appr. 3** ECC-Rep. CAM/2		**Appr. 4** ECC-Rep CAM/4		**Appr. 5** ECC-Rep CAM/6	
		%A	%P	%P	%A	%A	#It	%A	#It	%A	#It
10^{-4}	300	1.317	185.4	1.267	1.326	0.696	3.00	0.313	6.00	0.215	9.00
	1000	1.195	96.06	1.185	1.211	0.619	3.00	0.330	6.05	0.236	9.01
	3000	1.237	67.90	1.297	1.265	0.675	2.89	0.382	5.26	0.279	7.73
$3x\ 10^{-4}$	300	3.933	533.0	5.337	3.969	2.158	3.00	1.029	6.00	0.703	9.00
	1000	3.873	283.3	4.431	3.936	2.054	3.00	0.939	6.10	0.647	9.19
	3000	3.459	177.9	3.676	3.537	1.818	3.00	0.967	5.90	0.679	8.39
10^{-3}	300	12.75	1629	39.56	13.00	6.836	3.00	3.695	7.00	2.65	10.00
	1000	13.07	913.9	24.18	13.36	7.337	3.00	3.586	6.88	2.50	10.00
	3000	13.50	581.5	17.56	13.84	6.505	3.00	3.146	6.01	2.24	9.16

Dans le tableau V, la colonne 1 donne les densités des défauts P_f que nous avons expérimenté. Toutes les cas présentés concernent une capacité totale de 9,75 Gbit de SRAM, correspondant à un nombre total de 250M mots x 39 bits par mot (32 data bits et 7 Hamming code bits). La capacité totale de 9,75 Gbit est partitionnée en plusieurs mémoires embarquées distribuées dans un SoC. Nous avons considéré 3 cas pour cette distribution : 300 mémoires embarquées ; 1000 mémoires embarquées; and 3000 mémoires embarquées ; comme reporté dans la colonne 2 du tableau. Les colonnes 3 et 4 montrent les coûts en surface et en puissance dissipée (en pourcentage de la surface et de la puissance de la mémoire sous-réparation) pour la réparation conventionnelle.

Dans tous les cas du ECC-based repair, nous utilisons une CAM de réparation séparée de la CAM du diagnostic. Ainsi, comme pendant l'exécution de l'application nous utilisons uniquement la CAM de réparation qui est identique pour tous les cas de ECC-based repair, nous obtenons pour tous ces cas la même consommation de puissance lors de l'exécution des applications. Ainsi, la colonne 5 donne le coût en puissance dissipé pour tous les cas de ECC-based repair.

La colonne 6 donne le coût en surface pour l'approche utilisant une CAM de diagnostic de taille maximale (i.e. sans les réductions obtenues quand on utilise l'approche du diagnostic itératif). Les colonnes 7 et 8 donnent le coût en surface et le nombre moyen des itérations de test, lorsque nous utilisons une CAM de diagnostic ayant la moitié de la taille maximale. Les colonnes 9 et 10 donnent les mêmes paramètres lorsque la CAM du diagnostic est un quart de la taille maximale. Les colonnes 11 et 12 donnent les mêmes paramètres lorsque la CAM du diagnostic est un sixième de la taille maximale.

Dans le tableau V, nous observons que les coûts en surface et en puissance dissipée de la réparation conventionnelle (appelée «Appr 1» dans le tableau), augmentent à peu près linéairement avec la densité de défauts. Toutefois, le coût en surface est important mais pas énorme, alors que le coût en puissance dissipée devient excessif.

Notre approche utilisant une CAM de diagnostic séparée de la CAM de réparation (mentionnée comme «Appr. 1» dans le tableau), mais qui n'utilise pas l'approche du diagnostic itératif (ce qui maximise la taille de cette CAM), permet la réduction drastique de la consommation de puissance pendant l'exécution des applications en comparaison avec l'approche de réparation conventionnelle. Aussi, cette approche n'augmente pas la longueur du test (quoi que sa durée est quand même augmenté par cause de l'augmentation de la puissance dissipée pendant la phase de test et réparation), mais son coût en surface est élevé (similaire au cout de la réparation conventionnelle).

Le deuxième ensemble de nos approches (celles utilisant le diagnostic itératif – mentionnées «Appr. 3», «Appr. 4», et «Appr. 5» dans le tableau V), réalise la même réduction de la puissance dissipée pendant la phase d'exécution des applications que l'approche mentionnée comme «Appr. 1» dans le tableau V. En outre, il permet une réduction significative du coût en surface (en particulier pour les densités de défauts élevées). Cette amélioration se fait au détriment de la longueur de test, puisque l'algorithme de test devra être exécutée environ 3 fois pour le cas «ECC-repair CAM/2», 5 à 7 fois pour le cas «ECC-repair CAM/2», et 8 à 12 fois pour le cas «ECC-repair CAM/6», comme indiqué dans les colonnes étiquetées par #It, dont les résultats sont obtenus par le biais de notre algorithme d'injection des fautes et de pseudo-simulation. Notons cependant que, ces approches utilisent une CAM de diagnostic bien plus petite par rapport à l'approche de réparation conventionnelle («Appr. 1» dans le tableau V), et par rapport à l'approche de ECC-based repair utilisant un CAM de diagnostic séparée mais sans diagnostic itératif («Appr. 2» dans le tableau V). Ainsi, pendant la phase du test et de diagnostic elles auront une dissipation de puissance réduite.

Tableau VI. Augmentation de la durée du test

P_f	#Embedded Memories	Conventional Repair	**Appr. 3** ECC-Rep. CAM/2		**Appr. 4** ECC-Rep. CAM/4		**Appr. 5** ECC-Rep. CAM/6	
		Test Power	Test Power	Test-time increase	Test Power	Test-time increase	Test Power	Test-time increase
10^{-4}	300	2.85	1.99	**2.01**	1.47	**3.09**	1.33	**4.20**
	1000	1.96	1.51	**2.31**	1.27	**3.92**	1.19	**5.47**
	3000	1.68	1.36	**2.34**	1.18	**3.69**	1.13	**5.20**
$3x10^{-4}$	300	6.33	3.92	**1.86**	2.43	**2.30**	1.99	**2.83**
	1000	3.83	2.53	**1.98**	1.71	**2.72**	1.49	**3.57**
	3000	2.78	1.93	**2.01**	1.49	**3.16**	1.33	**4.01**
10^{-3}	300	17.29	9.74	**1.69**	5.69	**2.30**	4.32	**2.50**
	1000	10.14	6.01	**1.78**	3.44	**2.33**	2.69	**2.65**
	3000	6.81	4.07	**1.79**	2.47	**2.18**	2.01	**2.70**

Ce constat peut être exploitée afin de tester en parallèle dans le SoC un plus grand nombre de mémoires et de réduire l'impact sur la durée du test. Ainsi, comme nous pouvons constater dans le tableau VI, l'augmentation de la durée du test est plus petite que l'augmentation de sa longueur. En outre, comme le coût en surface augmente linéairement avec la densité des défauts et comme les mémoires embarquées occupent la plus grande partie des SoC modernes (plus de 90% de la surface dans la plupart des cas), le coût en surface des approches «Appr. 1» et «Appr. 2» devient tout à fait indésirable. Par exemple, pour la densité de défauts 10^{-3} le coût en surface est d'environ 13%, ce qui, dans un SoC dans lequel les mémoires embarquées occupent plus de 90% de sa surface, entraîne un coût supérieur à 11,7% de la surface totale du SoC. Ainsi, la réduction du le coût en surface obtenue par l'approche de diagnostic itérative est hautement souhaitable.

L'approche du diagnostic itératif (présentée dans cette section), complémente l'approche décrite dans la section précédente (utilisant des algorithmes de test du type SRDF), afin d'offrir au concepteur la possibilité de faire des compromis entre le coût de test, le coût en surface, et le coût en puissance, et pouvoir satisfaire au mieux ses contraintes en termes de ces coûts. En particulier nos résultats présentés dans les tableaux III, IV, V et VI, montrent que l'approche utilisant les algorithmes de test du type SRDF offre des meilleurs compromis des couts pour les plus grandes densités des défauts, tandis que l'approche utilisant le diagnostic itératif offre des meilleurs compromis des couts pour des densités des défauts plus modérées.

II.3 Architectures de Réparation à Basse Consommation

Les approches décrites dans les sections précédentes permettent des gains importants concernant les coûts en surface et en puissance dissipée. Néanmoins, malgré sa réduction drastique, le coût en surface et en consommation reste non négligeable. Ainsi, dans un contexte où la réduction de la puissance dissipée est très recherchée, il devient important de réduire encore plus la consommation de puissance de nos approches. Pour satisfaire ce prorogatif, nous proposons dans le chapitre 4 des architectures de réparation originales qui permettent une réduction encore plus substantielle de la puissance dissipée.

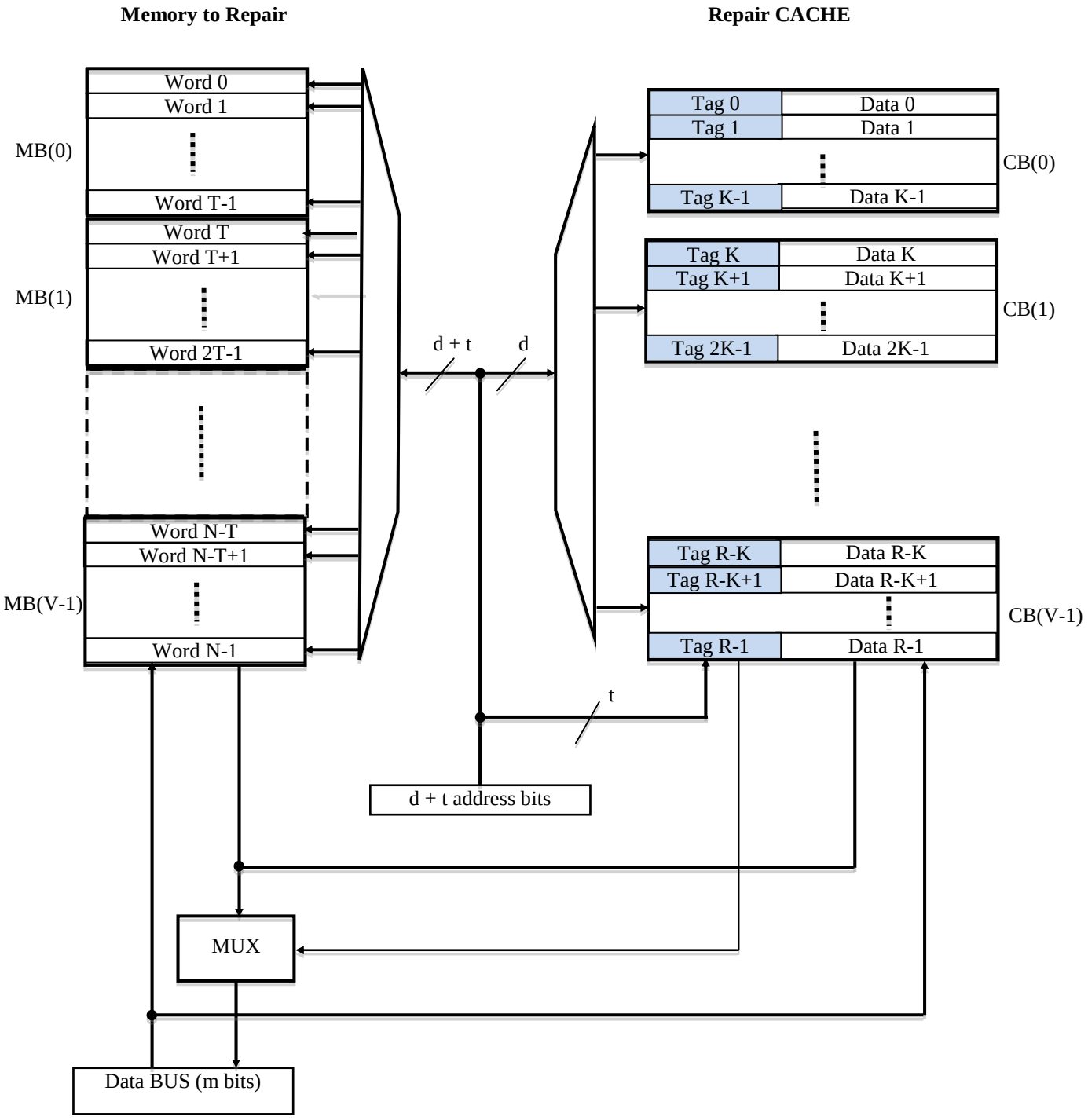


Figure V. Architecture de réparation à basse consommation utilisant une mémoire Cache.

Dans les architectures de réparation conventionnelles, les mots nécessitant réparation (i.e. les mots contenant plus qu'une cellule défaillante dans le cas du ECC-based repair) sont stockés dans une CAM, et lors de chaque accès mémoire l'adresse courant est comparée avec l'ensemble des adresses stockées dans la CAM. L'approche de ECC-based repair réduit drastiquement le nombre des mots stockés dans la CAM, mais pour des grandes densités de défauts ce nombre reste non-négligeable. Ainsi, à chaque accès mémoire un nombre de comparaisons non-négligeable est effectué, induisant une augmentation significative de la

puissance dissipée. Pour faire face à ce problème, nous avons proposé une architecture de réparation originale, qui partage l'espace d'adressage de la mémoire en plusieurs sous-espaces traités séparément. Ainsi, à chaque accès mémoire, l'adresse courant est comparée avec un sous-ensemble des adresses mémoire contenant des fautes multiples, réduisant de façon significative le nombre des comparaisons et la consommation dissipée correspondante. De cette façon, en utilisant un découpage très fin, la consommation de puissance est réduite de façon drastique. Cette architecture est présentée dans la figure V, et consiste en l'utilisation d'une mémoire CACHE (set-associative Cache), à la place de la CAM de réparation, et serra référée dans la suite comme *architecture à Cache*.

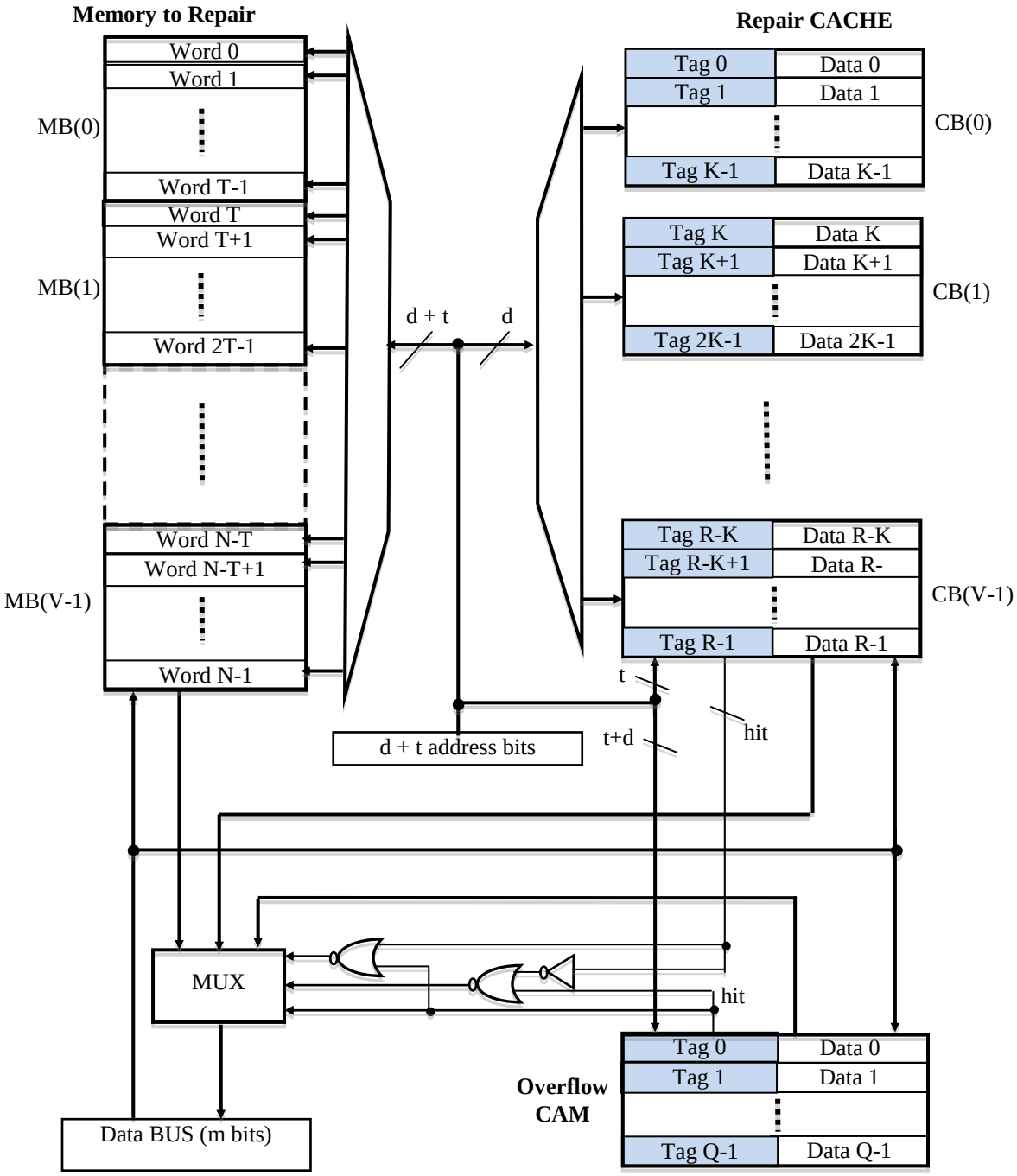


Figure VI. Architecture de réparation utilisant une CAM de débordement

Le découpage de l'espace d'adressage réalisé par l'architecture de la figure V, réduit la puissance dissipée de façon significative. Néanmoins, comme le montre notre analyse, un tel découpage pourra aussi augmenter de façon significative le coût en surface. Nous avons alors proposé une architecture à plusieurs niveaux, qui permet de réduire de façon drastique la consommation toute en maintenant un coût en surface faible. Cette architecture est montrée dans la figure VI. En effet, pour réparer la mémoire, l'architecture de la figure V doit réparer tous les sous-ensembles dans lesquelles on partitionne l'espace d'adressage de la mémoire. Comme la distribution des mots défaillantes n'est pas uniforme, certains de ces sous-ensembles contiendrons plus des mots défaillantes que d'autres. Comme nous ne savons pas au moment de la conception du système quels sous-ensembles contiendrons plus des fautes que d'autres, nous seront obligés d'augmenter le nombre des mots utilisés dans chaque *set* de la Cache. Ceci augmente le coût en surface, mais aussi accessoirement la puissance dissipée. Ces augmentations des coûts sont encore plus prononcées si nous augmentons le nombre des sous-ensembles de l'espace d'adressage, pour : réduire le nombre des mots de chaque *set* de la Cache afin de réduire lors de chaque accès mémoire le nombre des comparaisons d'adresses, et réduire ainsi la puissance dissipée. Ainsi, afin de réduire ces coûts, nous avons proposé l'architecture de réparation présentée dans la figure VI. Cette architecture réduit le nombre des mots utilisés dans chaque *set* de la Cache, et utilise une CAM de débordement (*architecture à CAM de débordement*) afin de réparer de façon complet les quelques sous-ensembles de l'espace d'adressage qui excédent la capacité des *sets* correspondants de la Cache.

Finalement, afin de réduire la puissance dissipée de la CAM de débordement, nous avons aussi proposé une variante de l'architecture de la figure VI, dans laquelle nous remplaçons la CAM de débordement par une Cache de débordement (*architecture à Cache de débordement*).

Les résultats de l'évaluation des architectures décrites précédemment sont présentés dans les tableaux VII et VIII. Le tableau VII présente l'évaluation des nouvelles architectures dans le contexte de réparation conventionnelle, tandis que le tableau VIII présente l'évaluation des nouvelles architectures dans le contexte de ECC-based repair. Dans les deux tableaux, la colonne 1 donne la densité des défauts. Toutes les cas présentés dans ces tableaux concernent une capacité totale de 9,75 Gbit de SRAM, correspondant à un nombre total de 250M mots x 39 bits par mot (32 data bits et 7 Hamming code bits). La capacité totale de 9,75 Gbit est partitionnée en plusieurs mémoires embarquées dans un SoC. Nous avons considéré 3 cas pour cette distribution : 300 mémoires embarquées; 1000 mémoires embarquées; et 3000 mémoires embarquées ; comme reporté dans la colonne 2 des deux tableaux. Aussi les résultats présentés dans ces tableaux sont obtenus pour une efficacité de réparation (rendement) de 90% pour la totalité de la capacité mémoire du SoC (i.e. pour 9,75 Gbit de SRAM).

Tableau VII. Evaluation des nouvelles architectures dans le contexte de réparation conventionnelle

P_f	#Emb Mem	Non-ECC Repair CAM			Non-ECC Repair CACHE-1 / CACHE-2					
		N_{CW}	%A	%P	N_{S1}	N_{W1}	N_{S2}	N_{W2}	%A	%P
10^{-4}	300	3466	1.32	185.3	64	63	2	39	1.879	22.73
	3000	402	1.27	67.90	32	18	**1**	1**3**	2.376	23.94
3x 10^{-4}	300	10285	3.93	532.9	128	99	2	30	4.548	33.23
	3000	1121	3.46	177.9	64	24	2	20	6.251	42.33
10^{-3}	300	35325	12.75	1629	512	85	64	32	15.20	65.16
	3000	3693	13.49	581.5	128	39	2	27	15.19	70.33

Dans le tableau VII, les colonnes 3, 4, et 5 donnent les résultats pour l'approche de réparation conventionnelle utilisant une CAM de réparation : la colonne 3 présente le nombre des mots CAM nécessaires pour obtenir le rendement de 90%, les colonnes 4 et 5 donnent les coûts en surface et en puissance dissipée. Les colonnes 6 à 11 donnent les résultats pour *l'architecture à Cache de débordement*, laquelle emploie deux Caches (CACHE 1 and CACHE 2): les colonnes 6 et 7 donnent le nombre des *sets* et le nombre des *ways* de la CACHE 1; les colonnes 8 et 9 donnent les mêmes paramètres pour la CACHE 2; les colonnes 10 et 11 donnent les coûts en surface et en puissance dissipée.

Dans le tableau VII, nous observons que pour l'approche de réparation conventionnelle, la nouvelle architecture permet une réduction drastique du coût de la puissance dissipée en contrepartie d'une légère augmentation du coût en surface.

Tableau VIII. Evaluation des nouvelles architectures dans le contexte de ECC-based repair.

P_f	#Emb Mem	ECC Repair CAM			ECC Repair CACHE-1 / CACHE-2					
		N_{CW}	%A	%P	N_{S1}	N_{W1}	N_{S2}	N_{W2}	%A	%P
10^{-4}	300	16	0.008	1.267	4	8	-	-	1.201	0.017
	3000	6	0.028	1.297	-	-	-	-	-	-
$3x10^{-4}$	300	83	0.036	5.337	16	6	1	12	0.069	3.723
	3000	17	0.078	3.676	-	-	-	-	-	-
10^{-3}	300	720	0.249	39.56	64	14	2	30	0.544	9.68
	3000	98	0.344	17.56	16	8	1	10	0.646	9.93

Dans le tableau VIII, qui présente les résultats des nouvelles architectures dans le contexte de ECC-based repair, nous ne donnons pas des résultats dans les cas où l'architecture traditionnelle utilise une CAM de réparation de petit taille (car dans ces cas les améliorations obtenues par les nouvelles architectures de réparation restent marginales). Nous observons que *l'architecture à Cache de débordement* (référée comme CACHE-1 / CACHE-2 dans le tableau VIII) permet dans la plupart des cas une réduction significative du coût en puissance dissipée. Ainsi, on obtient un coût en puissance dissipée faible pour les densités des défauts élevées ($P_f = 10^{-4}$ and $P_f = 3x10^{-4}$), et un coût en puissance dissipée modéré (moins de 10%) pour les densités des défauts très élevées ($P_f = 10^{-3}$). De plus, une technique prometteuse présentée dans le chapitre 4 - section 4.4 (mais pas encore évalué), devrait permettre une réduction supplémentaire de la puissance dissipée.

II.4 Mathématiques de Calcul du Rendement pour les Architectures de Réparation des Mémoires

Dans le chapitre 5 nous présentons les expressions analytiques et les algorithmes que nous avons développé afin de pouvoir calculer les rendements obtenus par nos nouvelles approches et architectures de réparation.

Nous pouvons calculer les rendements obtenus par l'approche de réparation conventionnelle ainsi que par l'approche de ECC-based repair en utilisant l'expression analytique suivante :

$$Y = \sum_{t=0}^{N_{wc}} \left(\frac{N_w! P_{wg}^{(N_W - t)}}{(N_w - t)!\, t!} (1 - P_{wg})^t \sum_{r=0}^{N_{wc}-t} \frac{N_{wc}! P_{wcg}^{N_{wc}-r}}{(N_{wc} - r)!\, r!} (1 - P_{wcg})^r \right) \quad (1).$$

Dans cette expression, N_W est le nombre des mots de la mémoire ; N_{WC} est le nombre des mots de la CAM ; P_{Wg} donne la probabilité qu'un mot mémoire n'a pas besoin d'être réparé (good word) ; P_{Wcg} donne la probabilité qu'un mot de la CAM peut être utilisé pour réparer un mot défaillant de la mémoire (good CAM word). Dans l'approche de réparation conventionnelle, Pwg est égal à la probabilité que le mot mémoire n'est pas défaillant et peut être calculé par $P_{Wg} = (1 - P_f)^N$, où N est le nombre des bits du mot mémoire et P_f est la probabilité qu'une cellule mémoire est défaillante. Dans l'approche de ECC-based repair P_{Wg} donne la probabilité que le mot mémoire n'est pas défaillant ou qu'il contient une cellule défaillante. Ainsi, dans cette approche, P_{Wg} peut être calculé par $P_{Wg} = (1 - P_f)^N + N(1 - P_f)^{N-1}P_f$. P_{Wcg} est calculé d'une façon similaire (voir chapitre 5).

Comme déterminé dans le chapitre 5, le nombre d'opérations nécessaire pour calculer le rendement au moyen de l'expression (1) est : $Nw(Nwc + 1) + (Nwc^2 - 1)(5Nwc + 12)/6 + 1$ multiplications; $(Nwc + 1)(Nwc + 4)/2$ divisions; et $N_{WC}(N_{WC} + 3)/2$ additions, où Nw est le nombre de mots de la mémoire, et Nwc est le nombre de mots de la CAM de réparation.

Pour les grandes densités des défauts, la complexité du calcul est beaucoup plus élevée par rapport au calcul du rendement pour des faibles densités de défauts, car, dans ce dernier cas, les défauts affectant la CAM ont un impact insignifiant sur le rendement et sont ignorés, ce qui donne une expression pour le calcul du rendement beaucoup plus simple. En outre, comme nous considérons des futures technologies très avancées, permettant la fabrication de puces très complexes, nous devons être en mesure de faire face à des mémoires de très grande taille. Aussi, comme nous considérons des grandes densités de défauts, nous dévons aussi être capables de traiter des grandes CAM de réparation. Dans ce contexte, les nombres des opérations donnés ci-dessus deviennent trop grands. De plus, ces opérations doivent manipuler de très grands nombres ainsi que de très petits nombres, nécessitant l'utilisation d'une arithmétique de haute précision. Ainsi, le calcul du rendement par le biais de l'expression (1) devient infaisable dans de temps réalistes. Pour accélérer ce calcul, nous avons découvert certaines relations récursives inédites, décrites dans le chapitre 5, qui réduisent le nombre d'opérations à une complexité linéaire, nécessitant seulement : $N_W + 8N_{WC} - 1$ multiplications, $2N_{WC}$ divisions, $2N_{WC}$ additions, et N_{WC} soustractions. Ces nombres d'opérations sont drastiquement plus petits que les nombres d'opérations nécessaires pour calculer l'expression (1) de manière directe.

La nouvelle approche de calcul de rendement récursif a été implémentée en C ++, et nous avons pu calculer dans des temps très courts les rendements de réparation pour les architectures utilisant une CAM de réparation.

L'expression (1) utilisée pour calculer le rendement dans le cas des architectures utilisant une CAM de réparation (*architecture à CAM*), peut être aussi utilisée dans le cas des architectures utilisant une Cache de réparation (*architecture à Cache*) montrée dans la figure V. En effet, dans ce cas, nous pouvons considérer que nous avons un système comportant M mémoires (où M est le nombre des sous-ensembles dans lesquelles est partitionné l'espace d'adressage de la mémoire), et chacune de ces mémoires est réparée par le *set* correspondant de la set-associative Cache. Ainsi, nous pouvons calculer le rendement Y pour chacune des M mémoires en la considérant comme une mémoire réparé par un CAM ayant un nombre des mots égal au nombre K des mots (*ways*) de chaque *set* de la set-associative Cache. Ensuite, le rendement global de la mémoire sera donnée par $Y_{MEM} = Y^M$.

Malheureusement le calcul du rendement pour *l'architecture à CAM de débordement* et nécessite le développement d'une nouvelle approche analytique. Cette approche pourra aussi être utilisée pour calculer le rendement de *l'architecture à Cache de débordement*, en considérant que nous avons un système comportant M mémoires (où M est le nombre des *sets* de la Cache de débordement), et en employant à chacune de ces mémoires l'approche du calcul du rendement développée pour *l'architecture à CAM de débordement*. Ainsi, dans la suite nous présentons l'approche de calcul du rendement pour *l'architecture à CAM de débordement*.

Soit N_S le nombre des *sets* (Set(1), Set(2), … Set(N_S)) de la set-associative Cache utilisée dans *l'architecture à CAM de débordement*. Dans cette architecture, la mémoire est partitionnée virtuellement dans N_S blocks MB(1), MB(2), … MB(N_S) réparés respectivement par les *sets* Set(1), Set(2), … Set(N_S) de la set-associative Cache.

Soit : N_{WS} le nombre des mots de chaque *set* de la set-associative Cache (i.e. le nombre des *ways* de cette Cache) ; N_{WB} le nombre des mots de chacun des N_S blocks de la mémoire ; N_d le nombre des bits des données de chaque mot de la mémoire (qui est aussi le nombre des bits des données de chaque mot de la set-associative Cache et de la CAM de débordement).

Soit N_{t1} le nombre des bits du champ de *tag* de la set-associative Cache; N_{t2} le nombre des bits du champ de *tag* de la CAM de débordement ; et N_f le nombre des *flag* bits des mots de la set-associative Cache et de la CAM de débordement.

Dans le ECC-based repair, la probabilité qu'un mot mémoire n'a pas besoin d'être réparé (good word) est égale à la probabilité que le mot n'est pas défaillant ou qu'il contient une cellule défaillante. Ainsi cette probabilité est donnée par :

$$P_{WMG} = (1 - P_f)^{Nd} + Nd(1 - P_f)^{Nd-1} P_f$$

Par des considérations similaires, et en considérant que la surface d'une cellule *tag* est q fois plus large que la surface de la cellule mémoire et que la surface d'une cellule *flag* est r fois plus large que la surface de la cellule mémoire, nous trouvons que la probabilité qu'un mot de la set-associative Cache peut être utilisé pour réparer un mot défaillant de la mémoire (good Cache word), est donnée par :

$$P_{WSG} = (1 - P_f)^{(qNtl + rNf)}((1 - P_f)^{Nd} + Nd(1 - P_f)^{Nd-1} P_f).$$

La probabilité qu'un mot de la CAM de débordement peut être utilisé pour réparer un mot défaillant de la mémoire (good CAM word), est donnée par :

$$P_{WOG} = (1 - P_f)^{(qNt2 + rNf)}((1 - P_f)^{Nd} + Nd(1 - P_f)^{Nd-1} P_f).$$

La probabilité qu'un *set* de la set-associative Cache répare tous les mots nécessitant réparation dans le block correspondant de la mémoire est :

$$P_{0UF} = \sum_{t=0}^{N_{WS}} \left(\frac{N_{WB}! P_{WMG}^{(N_{WB}-t)}}{(N_{WB}-t)!\,t!} (1 - P_{WMG})^t \sum_{r=0}^{N_{WS}-t} \frac{N_{WS}! P_{WSG}^{N_{WS}-r}}{(N_{WS}-r)!\,r!} (1 - P_{WSG})^r \right) \quad \textbf{(2)}$$

La probabilité qu'un *set* de la set-associative Cache laisse non-réparés exactement k mots nécessitant réparation dans le block correspondant de la mémoire est :

$$P_{kUF} = \sum_{t=k}^{N_{WS}+k} \left(\frac{N_{WB}! P_{WMG}^{(N_{WB}-t)}}{(N_{WB}-t)!\,t!} (1 - P_{WMG})^t \frac{N_{WS}! P_{WSG}^{(t-k)}}{(N_{WS}-t+k)!\,(t-k)!} (1 - P_{WSG})^{(N_{WS}-t+k)} \right) \quad \textbf{(3)}$$

Soit k(1), k(2), … k(N_S) le nombre des mots des N_S blocks MB(1), MB(2), …MB(N_S) de la mémoire qui sont laissés non-réparés respectivement par les *sets* Set(1), Set(2), … Set(N_S) de la set-associative Cache. La probabilité que k(1) mots de MB1 et k(2) mots de MB2 … et k(N_S) mots de MBN_S

sont laissés non-réparés est égale à: $P_{k(1)UF}P_{k(2)UF}\ldots P_{k(Ns)UF}$, où les valeurs des probabilités $P_{k(i)UF}$, sont calculés par l'expression (2) pour k(i) = 0 et par l'expression (3) pour k(i) > 0.
Pour réparer ces mots, la CAM de débordement doit disposer au moins Q = k(1) + k(2) + … $k(N_S)$ mots correctes (good words).
Soit N_{WO} le nombre des mots de la CAM de débordement. La probabilité que la CAM de débordement dispose au moins Q mots corrects est :

$$P_{QCO}=\sum_{u=0}^{N_{WO}-Q}\frac{N_{WO}!P_{WOG}^{(Nwo-u)}}{(N_{WO}-u)!u!}(1-P_{WOG})^{u} \qquad (4)$$

où $P_{WOG}=(1-P_f)^{(2.8Nt2+2Nf)}((1-P_f)^{Nd}+Nd(1-P_f)^{Nd-1}P_f)$ est la probabilité qu'un mot de la CAM de débordement es correcte, comme déterminer plutôt.
Alors, la probabilité que la mémoire est réparée quand k(1) mots de MB1, k(2) mots de MB2, … $k(N_S)$ mots de MBN_S (avec $0 \le k(i)\ \forall\ i \in \{0, 1, \ldots N_S\}$,) sont laissés non-réparés par la set-associative Cache est donnée par :

$PR_{k(1),k(2),\ldots k(Ns)} = P_{QCO}P_{k(1)UF}P_{k(2)UF}\ldots P_{k(Ns)UF}$ **(5)**

où k(1) + k(2) + … $k(N_S)$ = Q, et P_{QCO} est calculé par l'expression (4).
Pour Q = k(1) + k(2) + … $k(N_S) > N_{WO}$ nous avons $P_{QCO} = 0$. Ainsi, on doit considérer seulement les cas où Q = k(1) + k(2) + … $k(N_S) \le N_{WO}$. Par conséquent, pour calculer la probabilité totale que la mémoire soit réparée, nous devons prendre la somme des probabilités $PR_{k(1),k(2),\ldots k(Ns)}$ pour toutes les combinaisons des valeurs possibles de N_S entiers positifs k(1), k(2), … $k(N_S)$ dont la somme est égale à Q, et pour tout entier $Q \le N_{WO}$.

Dans la théorie des nombres, les combinaisons de Ns entiers positives ayant une somme égale à Q sont connues comme les compositions de Q dans N_S parts. Ce nombre de compositions est égal à $C'_{Ns(Q)} = (Q+ N_S-1)!/Q!(N_S-1)!$, et donne une valeur énorme dans la plupart des cas d'intérêt pratique dans le cadre de cette thèse. Par exemple, pour N_S = 64 et Q = 32 (qui sont nécessaires quand on utilise une set-associative Cache ayant 64 *sets* et 32 *ways*), nous trouvons un nombre énorme de compositions ($C'_{Ns(Q)} \approx 1{,}9801165182011\text{x}10^{25}$), qui ne permet pas de calculer toutes les probabilités $PR_{k(1),k(2),\ldots k(Ns)}$ correspondantes dans un temps de calcul réaliste. Ainsi, nous avons besoin d'une approche plus efficace. La solution à ce problème complexe est obtenue dans le chapitre 5 en exploitant le théorème des nombres pentagonaux de Leonard Euler. Le développement des algorithmes rapides de calcul du rendement pour *l'architecture à CAM de débordement*, utilisant cette solution, est aussi présenté dans le chapitre 5.

Ces algorithmes ont étaient été implémentée en C ++, et nous ont permit de calculer dans des temps courts les rendements de réparation pour *l'architecture à CAM de débordement* ainsi que pour *l'architecture à Cache de débordement*.

II.5 BIST Transparent pour ECC-Based Memory Repair

Comme le test des mémoires effectue des opérations de lecture et d'écriture sur tous les mots de la mémoire, il détruit leur contenu. Ainsi, il n'est pas permissible de tester une mémoire pendant l'exécution d'une application car il détruirait le contexte de l'application. Le BIST transparent des mémoires a était proposé afin de faire face à cette contrainte. Cette technique utilise comme données de test le contenu des mots de la mémoire en les transformant de manière réversible de façon à restaurer leur contenu à la fin du test.

Les prévisions estiment que les technologies CMOS ultimes et post-CMOS devraient accélérer le vieillissement des circuits, entraînant une augmentation importante de la fréquence d'occurrence des fautes. Ainsi, des sessions de test de mémoire fréquentes doivent être activées même lors de l'exécution des

applications. Ainsi, ces tests exigent l'utilisation de l'approche du BIST transparent. Mais cette approche utilise l'analyse de signature pour vérifier les données lues dans la mémoire pendant la phase du test. Cependant, l'analyse de signature ne peut pas distinguer les mots de mémoire comprenant une cellule défectueuse de ceux comprenant plusieurs cellules défectueuses, comme l'exige l'approche de ECC-based repair. Ainsi, les techniques de BIST transparent existantes ne sont pas compatibles avec le ECC-based repair. Une possibilité qui pourrait être utilisée pour résoudre ce problème serait d'utiliser le code ECC pour vérifier les données lues dans la mémoire pendant la phase du test. Mais comme le code ECC peut de façon incorrecte identifier des mots contenant des erreurs multiples pour des mots contenant des erreurs simples, et parfois pour des mots corrects, cette approche ne permet pas de résoudre le problème.

Pour faire face à ces difficultés, le chapitre 6 propose une approche de diagnostic hybride pour BIST transparent, capable de diagnostiquer l'existence de mots mémoire contenant plusieurs cellules défaillantes. Dans le chapitre 6 nous avons en effet proposé une variante de cette approche compatible avec l'approche de ECC-based repair utilisant les algorithmes de test du type SRDF, et une deuxième variante compatible avec l'approche de ECC-based repair utilisant une CAM dédiée au diagnostic.

Pour rentre compatible le BIST transparent avec l'approche du ECC-based repair nous devons s'assurer que : en utilisant l'analyse de signature et le code ECC comme moyens de vérification des données lue par la pendant l'exécution de l'algorithme de test, nous serons capables de déterminer si la mémoire contient des mots comportant plus d'une cellule défaillante. Mais comme mentionné précédemment, l'analyse de signature ne peut pas distinguer les mots de mémoire comprenant une cellule défectueuse de ceux comprenant plusieurs cellules défectueuses, et de l'autre coté, le code ECC peut identifier des mots contenant plus que deux erreurs comme des mots contenant des erreurs simples, ou pour des mots corrects. Ainsi, l'analyse de signature et le code ECC peuvent produire un faut diagnostic. Pour relever ce problème nous avons proposé deux principes illustrés dans la figure VII :

Principe 1 : Au lieu d'injecter dans l'analyseur de signature les données brutes lues dans la mémoire nous injecterons les données corrigées par le code ECC. Comme le ECC corrige toutes les erreurs à multiplicité égale à 1, si lors de l'exécution du test il n'y a pas des données erronées ou si tous les données erronées contiennent au maximum une erreur, la signature serra correcte. De l'autre coté, comme le ECC ne peut pas corriger des erreurs à multiplicité supérieure à 1, si lors de l'exécution du test il y a des données erronées contenant plus d'une erreur la signature serra erronée.

Principe 2 : Si l'approche de ECC-based repair utilise une CAM de diagnostic, alors, au lieu d'écrire dans cette CAM les données brutes lues dans la mémoire nous écrirons leurs syndromes calculés par le code ECC. Comme le ECC identifie correctement les positions des erreurs quand un mot contient une seul erreur, ce principe écrira dans la CAM du diagnostic les positions correctes des erreurs si un mot lues dans la mémoire contient une ou aucune erreur.

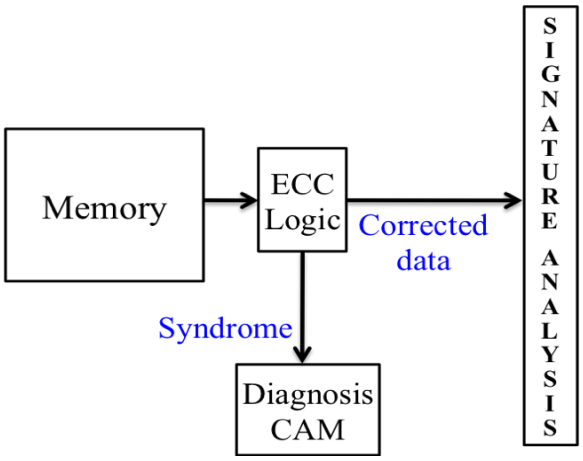


Figure VII. Principes permettant de rentre compatible le BIST transparent avec le ECC-based repair

COMPATIBILITE AVEC L'APPROCHE DE ECC-BASED REPAIR UTILISANT DES ALGORITHMES DE TEST DU TYPE SRDF

En utilisant le principe 1, nous pouvons rendre compatible le BIST transparent avec l'approche de ECC-based repair utilisant des algorithmes de test du type SRDF, comme illustré dans la figure VIII. En effet :

- Si la mémoire ne contient pas des mots comportant plus d'une cellule défaillante, aucun mot lu lors du test ne contiendra plus d'une erreur. Ainsi, selon le principe 1, l'injection dans l'analyseur de signature des données corrigées produira une signature correcte, et la mémoire sera correctement diagnostiquée comme ne contenant aucun mot comportant plus d'une cellule défaillante.
- Si la mémoire contient des mots comportant plus d'une cellule défaillante, alors pour chacun de ces mots l'algorithme de test du type SRDF garantira qu'il y aura au moins une opération de lecture de l'algorithme qui produira au moins deux erreurs. Dans ce cas : si une de ces lectures produit une erreur double, le signal de détection d'erreur du code ECC va signaler cette erreur double, et la mémoire sera correctement diagnostiquée comme contenant des mots comportant plus d'une cellule défaillante. Si toutes ces lectures produisent des erreurs de multiplicité supérieure à deux, selon le principe 1 l'injection dans l'analyseur de signature des données corrigées produira une signature erronée, et la mémoire sera correctement diagnostiquée comme contenant des mots comportant plus d'une cellule défaillante.

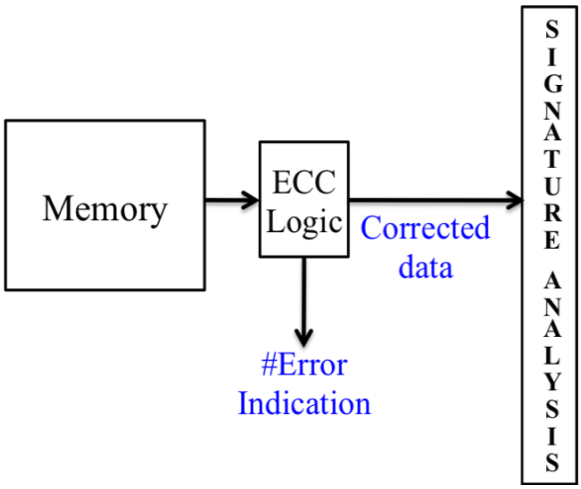


Figure VIII. Principe utilisée dans le BIST transparent compatible avec le ECC-based repair utilisant des algorithmes de test du type SRDF.

COMPATIBILITE AVEC L'APPROCHE DE ECC-BASED REPAIR UTILISANT UNE CAM DE DIAGNOSTIC

En utilisant le principe 2, nous pouvons rendre compatible le BIST transparent avec l'approche de ECC-based repair utilisant une CAM de diagnostic, comme illustré dans la figure IX. En effet :

- Si une opération de lecture de l'algorithme de test produit une erreur double, le signal de détection d'erreur du code ECC va signaler cette erreur double, et la mémoire sera correctement diagnostiquée comme contenant des mots comportant plus d'une cellule défaillante.
- Si une ou plusieurs opérations de lecture de l'algorithme de test produisent des erreurs de multiplicité supérieure à deux, selon le principe 1 l'injection dans l'analyseur de signature des données corrigées produira une signature erronée, et la mémoire sera correctement diagnostiquée comme contenant des mots comportant plus d'une cellule défaillante.
- Si aucune opération de lecture de l'algorithme de test ne produit plus d'une erreur, l'analyseur de signature produira une signature correcte, même si la mémoire contient des mots comportant plus d'une cellule défaillante, Ainsi, dans ce cas l'analyse de signature ne permettra un diagnostique correcte si la mémoire contient des mots comportant plus d'une cellule défaillante. Néanmoins, l'écriture dans la CAM du diagnostic des syndromes calculés par le code ECC selon le principe 2, garantira que la CAM mémorisera les positions correctes des erreurs. Ainsi, le contenu de la CAM permettra de diagnostiquer correctement la mémoire quel qu'il soit le cas (i.e. qu'elle contient ou qu'elle ne contient pas des mots comportant plus d'une cellule défaillante). Notons que s'il y a des mots comportant plus d'une cellule défaillante, l'utilisation d'un algorithme de test du type SRDF garantie l'existence d'opérations des lectures détectant au moins deux erreurs. Ainsi, l'emploie de ces algorithmes garanti un diagnostic correct par la seule utilisation du principe 1. Mais ceci n'est pas le cas pour les algorithmes conventionnels qui sont employés quand on utilise une CAM de diagnostic. Ainsi dans ce cas il était nécessaire d'utiliser aussi le principe 2.

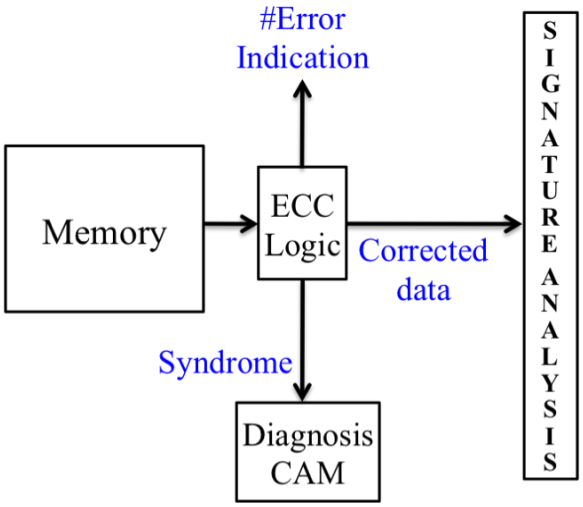


Figure IX. Principe utilisée dans le BIST transparent compatible avec le ECC-based repair utilisant une CAM de diagnostic.

III Conclusion

L'objectif de cette thèse concerne la réparation des mémoires affectées par des grandes densités des défauts. La seule technique réaliste pour la réparation des mémoires affectées par des grandes densités des défauts (connue comme «ECC-based memory repair»), mais dans le chapitre 1 de cette thèse nous montrons que cette technique souffrait d'un handicap de taille : pour des grandes densités des défauts les circuits de diagnostic des fautes deviennent très complexes et ajoutent un coût matériel très élevé, qui, malgré le faible coût en circuit de réparation de la technique «ECC-based memory repair», induit un coût total très élevé. Dans le chapitre 1 nous montrons aussi que malgré la réduction drastique des coûts en surface et en consommation de puissance par rapport aux techniques classiques, le cout en consommation de puissance reste non négligeable. Pour remédier à ces problèmes nous avons développé une panoplie de solutions.

La première solution, décrite dans le 2ème chapitre, propose une nouvelle famille d'algorithmes de test permettant d'éliminer complètement la circuiterie de diagnostic des fautes. En éliminant complètement les circuits de diagnostic, ces algorithmes permettent l'implémentation de l'approche «ECC-based memory repair» à coût minimal, ce qui représente un avantage décisif. Le seul inconvénient de cette approche est une augmentation sensible de la durée du test.

Afin de proposer une solution de diagnostique alternative, le 3ème chapitre propose une architecture de diagnostic itératif, qui permet des compromis entre les coûts en surface et les durées du test. Ainsi, en combinaison avec l'approche des algorithmes de test développés au deuxième chapitre, le concepteur dispose une panoplie de solutions permettant un vaste champ de compromis (partant d'un coût en surface minimal et un coût en durée de test maximal, à un coût en surface maximal et un coût en durée de test minimal).

Afin de réduire aussi la puissance consommée, le 4ème chapitre propose une architecture de réparation innovante, qui réduit de façon drastique la puissance de la circuiterie de réparation. La nouvelle

architecture utilise une combinaison d'une mémoire Cache associative et d'une mémoire CAM de très petite taille, dite de débordement, qui permet de réparer le faible nombre des mots mémoires qui risquent d'être laissées irréparables par la mémoire Cache. Grace à cette architecture le coût de consommation est réduit drastiquement, tandis que le cout en surface reste toujours très faible.

La considération des mémoires de très grand taille, des grandes densités des défauts et des architectures de réparation combinant des codes ECC, des CAMs, et des Caches, rend le calcul du rendement infaisable dans des temps réalistes. Pour remédier à ce problème, le 5ème chapitre propose des nouvelles approches mathématiques pour le calcul du rendement, qui réduit la complexité de ce calcul d'un grand nombre d'ordres de grandeur et permet de calculer le rendement en des temps très courts.

Les derniers développements de cette thèse concernent l'utilisation de l'approche de «ECC-based repair», pour détecter et réparer les fautes en phase d'utilisation du circuit. Ceci devient possible par l'approche dite de BIST transparent, qui teste une mémoire sans altérer son contenu. Cette approche semblait incompatible avec le «ECC-based repair». Pour contourner ce problème le 6ème chapitre propose une injection astucieuse des syndromes du code correcteur dans le CAM du diagnostique et les données corrigées par le ECC dans l'analyseur de signature, et aboutit à une architecture de BIST transparent compatible avec l'approche de «ECC-based repair».

Ainsi, les travaux présentés dans cette thèse aboutissent pour la première fois à une plateforme permettant la réparation des mémoires pour des grandes densités des défauts à des faibles couts de surface et puissance, et ouvrent la voie à une miniaturisation agressive, laquelle, sans ce type d'approche, serra prématurément bloquée pour des raisons de rendement, de fiabilité, et de puissance dissipée.

CONTENTS

CHAPTER 1

INTRODUCTION

Embedded memories occupy the largest part of modern SoCs and include even larger proportions of transistors. As memories are designed very tightly to the technology limits, they are more prone to failures than other circuits. Thus, they concentrate the large majority of fabrication defects affecting yield adversely. Hence, memory Built-In Self-Repair [1-10][54-55] became early mandatory for maintaining acceptable fabrication yield. Also, it was early predicted [11] that achieving acceptable reliability levels is one of the most critical issues for late CMOS. Indeed, soft-errors caused by neutrons and alpha particles and other field failures are critical concerns in memories. Thus, during the last decade, ECC became mandatory for maintaining acceptable reliability levels [12-18].

Late-CMOS and beyond-CMOS technologies hold the promise of integrating trillions of devices in a single chip, enabling unprecedented computing power and may have profound impact on all computer application domains (embedded systems, telecommunication networks, internet infrastructure, cloud computing, …), and ultimately on science, technology and the society as a whole. These benefits will become reality if we are able to aggressively push technology scaling towards extremely small feature dimensions. However, technology scaling has adverse impact on: (1) *process*, *voltage* and *temperature* (*PVT*) variations; (2) sensitivity to radiations; (3) circuit aging and wearout [14]; and (4) power dissipation and thermal constraints. The resulting high defect levels, heterogeneous behavior of identical designs, circuit degradation over time, and integrated circuits complexity, affect fabrication yield and make the production of chips with acceptable yield, reliability, and power density, increasingly challenging. These issues are exacerbated as we move towards the ultimate CMOS and beyond-CMOS technologies, and, whatever are the future improvements of fabrication technologies, sooner or later they will stop the technology scaling and its beneficial impact on most other technological domains and economic activities. Thus, disposing techniques able to cope with as high fault rates as possible, is increasingly suitable as it will allow pushing the limits of technology scaling much farther than what could be done by the sole improvement of fabrication technologies.

The aim of this thesis is to provide a framework enabling low-cost memory repair for very high defect densities, and exploit it for ensuring high yield and high reliability in the context of: very high fabrication-fault rates induced by fabrication defects and exacerbated process variations; high field-failure rates produced by exacerbated circuit degradation over time due to aging, very-low voltage operation, and also soft errors; and at the same time meet stringent low-power constraints. Our work is developed in the context of *Cells* [23-26]: a framework addressing the design of high yield, high reliability, and low-power tera-device systems affected by high-defect densities. This is a non-trivial task as fault-tolerance induces

both: high area penalty (drastically reducing the available computing resources), and high power penalty (which is incompatible with stringent low power constraints). Furthermore, even the most robust fault-tolerant techniques using massive redundancy (e.g. duplication or TMR) do not work under high fault rates, where several copies of a TMR or duplicated component can be defective. To address this challenge, *Cells* employs innovative approaches at all levels of system design, enabling the mitigation of very high defect densities and the reduction of power dissipation at low cost.

In this challenging context, our work concerns the development of an important component of *Cells*, addressing the design of robust memories. To address this challenge, we developed a low-cost memory repair framework for very high defect densities, as well as a simple DfX strategy described in section 1.1, which employs this memory repair framework for achieving high yield, high reliability, and low power dissipation at low cost. This memory repair framework is based on the so-called ECC-based repair [19]. This scheme, which is based on the combination of ECC codes with word repair, is highly efficient for repairing memories affected by high defect densities:

- As shown in [19], combining word repair and ECC is the only cost-effective solution (in terms of area cost) for achieving acceptable fabrication yield for high defect rates.
- As stated in [20] and confirmed in [21] "the number of repairable faults dramatically increases by combining the ECC and redundancy techniques together".

However, as we show in section 1.2, previous work on ECC-based memory repair has neglected one important issue: the hardware cost reduction gained by using ECC-based repair can be lost due to diagnosis requirements. To cope with this problem, in this manuscript we propose and develop various solutions for tackling this task at low area and power cost, as well as new word-repair architectures for further reducing power dissipation, as well as new yield computation mathematics and related algorithms for tackling the yield computation complexity related with very large memory systems and very high fault rates.

1.1 DFX REUSE STRATEGY

During the last decade, soft-errors caused by atmospheric neutrons and alpha particles became a major reliability concern in modern electronic systems [12][13][22]. This trend has led to the systematic protection of memories by means of ECC, most often implemented as single-error correction - double-error detection (SEC-DED) codes. As process scaling has increased the susceptibility to multi-cell upsets MCUs), that outperform the capabilities of SEC-DED codes, interleaving is also used to guaranty that at most one cell upset can affect the same memory word (single-bit-upset – SBU). The typical soft-error rate (SER) per Mbit SRAM at 28nm planar bulk CMOS is about 180 FIT for single-cell upsets (SCUs) and 10 FIT for MCUs. The introduction of finFET is accompanied by sharp SER reduction (SER in 14nm finFET process is about 12 FIT for SCUs, and 0,6 FIT for MCUs), but memory soft-error protection remains necessary as increasing chip complexities maintain significant SER at chip level. Process scaling is accompanied by sharp critical charge reduction, increasing soft error sensitivity, but this trend is counterbalanced by the reduction of sensitive volume. Thus, the SER per Mbit SRAM in future finFET nodes is expected to be constant or slightly reduced. However, the SER per chip will be increased significantly, making memory soft-error mitigation increasingly mandatory.

As DfX techniques are proliferating (Design for Test, Design for Debug, Design for Yield, Design for Low-Power, Design for Reliability …), the amount of hardware dedicated to non-computational purposes grows substantially. It is therefore mandatory to combine different DfX techniques to moderate their impact on area, power and/or performance. As ECC is already implemented in memories for mitigating soft-errors, it may be reused to reduce the cost of ECC-based repair. The critical issue for this kind of reuse is however the potential reduction of soft-error mitigation efficiency. *As words containing one faulty cell are not repaired in ECC based repair, upsets affecting such words may invalidate SEC-DED protection*. In this work we consider high fault rates ranging from 10^{-5} to 10^{-3} probability for a memory cell to be faulty. In the

less favourable case (10^{-3} faulty-cell probability), corresponding to 2 or 3 orders of magnitude higher faulty-cell rates than in current technologies, considering memory words of 32 data bits and 7 SEC-DED check bits, gives a probability for a memory word to contain one faulty cell equal to $39x10^{-3}x0.999^{38}$. In a very large SRAM of 100 Gbit capacity, this gives an average of $39x10^{-3}x0.999^{38}x10^{11}/39$ words containing a single faulty cell. As the SER per Mbit is slightly reduced with process scaling but MCU rate increases, considering 8 FIT for SCUs and 2 FIT for MCUs per Mbit SRAM after several finFET process generations is a reasonable projection. Considering a mean number of 6 cells hit by an MCU and also that interleaving is employed as usually to ensure that an MCU affects only a single cell of the same memory word, we find that the per year rate of single-cell upsets affecting a memory word containing a single faulty cell is equal to $(0.999^{38}x10^{-3}x39x10^{11}/39)(8+2x6)(38/10^{6})(10^{-9}x365.25x24) = 0.641$, where: the first parenthesis gives the average number of words in the 100 Gbit SRAM containing one faulty cell; the second parenthesis gives the mean number of cells in a 1 Mbit SRAM affected by upsets in 10^{9} hours (1 FIT = 1 event in 10^{9} hours), considering 8 FIT per Mbit for SCUs, 2 FIT per Mbit for MCUs, and a mean of 6 cells affected by an MCU; the third parenthesis gives the number of fault free cells of a word containing one faulty cell (as only hits affecting a fault-free cell of the word can produce double errors), divided by the number of cells of a 1 Mbit memory; and the fourth parenthesis gives the number of events per year per FIT. Hence, for a very large memory (100 Gbit capacity), 0.641 events per year can produce a double error, which is not corrected but it is detected by the ECC. Thus, it can be fixed by check-point based rollback recovery employed in most cross-layer reliability approaches under development for addressing the reliability of upcoming process generations. Therefore, in the worst-case scenario of a 10^{-3} faulty-cell probability, the reuse of the ECC for ECC-based repair in a large SRAM of 100 Gbit capacity will induce about two error-recovery interruptions at every 3 years, which is insignificant. If we consider a still high but less excessive faulty-cell probability equal to 10^{-4}, reusing ECC for ECC-based repair in a 100 Gbit SRAM will induce one error-recovery interruption at every 15 years. In a future single-chip massively-parallel tera-device processor consisting in 4000 processing nodes using 250 Mbit memory per node (a total of 1 Terabit memory), and employing ECC-based repair as envisioned in the *CELLS* framework [23-26], for the 10^{-3} faulty-cell probability each node will experience 1 interruptions for error recovery at every 625 years, while for the 10^{-4} faulty-cell probability each node will experience 1 interruptions for error recovery at every 6000 years. Furthermore, as the *CELLS* framework performs check-point-free error recovery by means of an innovative approach exploiting hierarchical task allocation in the multiprocessor grid [27], performance lost induced by check-pointing is also eliminated. Thus, ECC-based repair reusing ECC implemented for soft-error mitigation is a winning strategy in all above scenarios.

Due to the aggressive process scaling, ultimate CMOS and post-CMOS technologies are expected to be affected by high densities of manufacturing and aging-induced faults, as well as high power and temperature densities. Thus, they will require powerful techniques for off-line repair, runtime fault tolerance, and low power. Our developments of efficient memory repair for high fault-densities are motivated by the goal to reuse the hardware resources of ECC-based repair in order to address all the above critical issues. First, the expectation of steadily worsening process variability and aging-induced circuit degradation in upcoming process nodes makes mandatory the development of efficient memory self-repair for high fault-densities, in order to improve manufacturing yield after fabrication and subsequently extend the life of the circuit by repairing aging-induced faults. Furthermore, disposing a self-repair technique for fault densities much higher than those required for manufacturing and aging-induced faults, will also allow drastically reducing power dissipation. Indeed, in this case we can reduce aggressively the operating voltage (Vdd) in order to achieve drastic reduction of power dissipation (i.e proportional to the square of the reduction of Vdd), and repair the memory cells that exhibit faulty behavior due to the reduced Vdd. Ultimately, reliability issues can also be caused by aging-induced faults that occur during the execution of an application. Such faults may affect the correct execution of the application during the time that elapses between their instant of

occurrence and the next test and repair session. Disposing a self-repair technique for fault densities even higher than those required for the three classes of faults considered above, will allow solving this reliability issue by testing the memory under more stringent conditions (lower voltage and/or higher speed) than the worst conditions used during the execution of any application. This is because such tests can detect and proactively repair those memory cells that do not yet exhibit faulty behavior, but are already weakened and have increased chances to become faulty before the next test and repair session.

1.2 AREA, POWER, AND DIAGNOSIS ISSUES

Area and Power Issues

RAMs are repaired by replacing faulty regular units by fault-free spare units. The resulting cost corresponds to the cost of the spare units and of the circuitry used to control the unit replacement. For low defect densities, the principal cost source is the spare units. However, in some repair schemes, as defect densities increase the cost of the circuitry controlling the replacement can become more important. Indeed, there are two schemes for replacing regular units by spare units.

The first stores into dedicated registers the positions of the faulty units gathered during the test phase. Then, dedicated circuitry decodes this information to generate the control signals of a set of MUXes used to disconnect faulty regular units and connect in their place fault-free spare units. However, for high defect densities, *we need to use a large number of small spare units to repair an even larger number of small regular units* [4]. As both the number R of regular units and the number S of spare units increase, the complexity of the hardware implementing the MUXes increases exponentially, since we need R MUXes of S+1 inputs and 1 output each. The circuitry controlling the MUXes also increases exponentially as it has to generate Rx(S+1) signals. In addition, the hardware cost increases more rapidly than the exponential increase of the number Rx(S+1) of the generated signals, as the complexity of the combinational function generating each of these signals is not constant but increases as Rx(S+1) increases. Thus, the exponential increase of the hardware cost and power dissipation is of higher order than Rx(S+1), and MUX-based repair is not scalable to high defect densities.

The second scheme uses a CAM for storing the addresses and data of faulty units. This approach is scalable in size for increasing defect densities, as the CAM size is proportional to the number of faulty units, resulting in linear area increase. Thus, CAM based repair is preferable for high defect densities. However, another critical issue is power, as CAMs are power hungry. Using ECC-based repair reduces drastically the size of the CAM (as we only repair words containing more than one faulty cells), resulting in drastic reduction of area and even more drastic reduction of power penalties.

Diagnosis Issues

For implementing ECC-based memory repair it is necessary to identify after fabrication the words that comprise more than one faulty cells. It is also important to identify such words in the field, in order to cope with faults occurring during circuit life due to aging-induced circuit degradation. However, if a word contains more than one faulty cells, then, memory test algorithms do not guaranty that all of them are detected within the same read operation. Indeed, one fault can be detected by a read belonging to a march element and another fault can be detected by a read belonging to another march element. Thus, with existing memory test algorithms, each time a read detects a faulty cell we need to store the address of the faulty word as well as the position(s) of the faulty cell(s), and then, use an algorithm for determining the words containing several faulty cells detected at different instants of the test algorithm. The most efficient solution

[4] The probability of a unit to be faulty is proportional to its size. Thus, large units have large probability to be faulty. In high defect densities, to avoid that most of the spare and regular units are faulty, we must use small spare and regular units. Thus, we have a very large number of regular units (the size of the memory divided by the size of a regular unit). We also have to use a large number of spare units (large numbers of faults should be repaired in high defect densities).

for storing this information and identifying the words containing multiple faulty cells is to use a CAM in which we store the faulty addresses in the tag-fields and the positions of the faulty bits in the associated data fields. As at the instant we detect one faulty cell in a memory word we don't know if this word contains also other faulty cells that will be detected later, then, each time we detect a faulty word we are obliged to store it in the CAM, whether it contains one or more faulty cells. Thus, we need a CAM of the same size as in the case where we do not exploit ECC for repair purposes. Therefore the benefits expected by the use of ECC-based repair are lost due to diagnosis requirements. Of course, it is possible to eliminate the diagnosis CAM if we shift out of the chip the faulty addresses and faulty cell positions and use the intelligence of an external tester for performing the diagnosis. However, current and future chip complexities make this approach inefficient. Furthermore, using an external tester will prevent test and repair for field failures, which is necessary as the rate of aging-induced faults is becoming high in advanced process nodes.

Proposed Schemes for Power and Area reduction

As we have illustrated above, diagnosis requirements in ECC-based memory repair imply using a CAM of the same size as in conventional (i.e. non-ECC-based repair). In high defect densities, the area and power cost of this CAM will be very high. To cope with these issues, we developed various schemes:

- The first scheme eliminates altogether the need of diagnosis hardware by introducing a new kind of test algorithms. These algorithms have the property that if a memory word contains two or more faulty cells, there is a read operation in the algorithm that detects at least two of them: *single-read double-fault detection* (SRDF) property. Thus, we store a memory word in the CAM only when a single read detects two or more faulty cells in this word. Hence, no memory word containing only one faulty cell is stored. As a consequence, there is no CAM size increase due to diagnosis issues. Algorithms exhibiting this property will be referred as SRDF test algorithms.
- The second scheme employs two separate CAMs. A large diagnosis CAM is used during each test and diagnosis session to determine the memory words that comprise multiple faulty cells, and at the end of this session the memory words are transferred to a second small CAM (the runtime-repair CAM). At run-time, only the latter CAM is used and powered. As it is drastically smaller than the diagnosis CAM (which has the same size as the CAM used in conventional repair), runtime power is drastically lower than in conventional Built-Self-Repair.
- In the third scheme, to reduce the high hardware cost of the diagnosis CAM used in the second approach, we employ a smaller diagnosis CAM, which could not store the addresses of all faulty memory words. To compensate the missing CAM capacity, we execute the test algorithm several times. After each iteration of the test algorithm, we free a part of the CAM to create space for treating new faults. This process could reduce fault coverage. A dedicated iterative diagnosis algorithm was developed to cope with this issue.

Both the SRDF algorithms and the iterative-test-and-diagnosis approach reduce hardware cost at the expense of test length. The test length in the case of the SRDF algorithms is constant regardless of the kind of faults and the failure rate of the target process. On the other hand, for a given hardware cost (i.e. a given CAM size), the number of test iterations used by the second approach (and thus the test time) depends on the number of fault detections occurring during the test algorithm. It also depends on the type of faults as the number of detection instances may differ from one fault type to another. For instance, in the March SS algorithm [28]: ({ ⇳(W0) ; ⇑(R0, R0, W0, R0, W1) ; ⇑ (R1, R1, W1, R1, W0) ; ⇓(R0, R0, W0, R0, W1) ; ⇓(R1, R1, W1, R1, W0) ; ⇳ (R0) }), a state fault SF < 0/1/- > affecting a memory cell will be detected at 7 different instances of the algorithm. Other faults are detected in a different number of instances. Thus, test length increases with the increase of the fault rates and may also increase for certain fault types. As a matter of fact, the two approaches are complementary and offer to the designer efficient trade-offs in terms of test length and hardware cost. The iterative-test-and-diagnosis approach could be used for lower fault rates and

after a certain level of fault rates (depending on memory size and target yield) the approach using SRDF test algorithms could be selected as more efficient. Thus, the interest of the SRDF test algorithms increases as we move towards ultimate CMOS and post-CMOS technologies, which are expected to be affected by high defect densities.

The rest of this manuscript is organized in the following manner.

Chapter 2 addresses the highly challenging task related with the development of SRDF test algorithms, which, due to the SRDF constraint, are substantially more difficult to develop with respect to any existing test algorithm.

Chapter 3 addresses the separate-CAM scheme and proposes an iterative diagnosis algorithm enabling trade-offs in terms of test duration and hardware cost. It also proposes a pseudo-simulation approach that accelerates drastically the fault injection experiments required for evaluating the new iterative diagnosis algorithms.

Chapter 4 presents and evaluates new word repair architectures, which reduce drastically runtime power dissipation.

Chapter 5 addresses the yield evaluation issue, which is becoming very complex due to the consideration of very large memories, very high defect densities, and sophisticated repair architectures. To address this challenge, chapter 5 presents new yield computation mathematics and the related yield computation algorithms, which were used in the previous sections for evaluating the different solutions proposed in these chapters.

Chapter 6 addresses testing the memories during application execution by preserving their content. So, it proposes a hybrid diagnosis scheme which combines signature analysis with ECC error detection and correction and allows the smooth cooperation between the transparent BIST and ECC-based repair.

CHAPTER 2

MEMORY TEST ALGORITHMS FOR ECC-BASED REPAIR

As highlighted in the previous chapter, the benefits gained by using ECC-based repair can be lost due to diagnosis issues. Here we address the development of SRDF test algorithms enabling eliminating this issue. These algorithms guaranty the *single-read double fault detection* property. That is: if a memory word contains two or more faults, there is a read operation in the algorithm that detects at least two of them. We first treat the case of words affected by two faults (propositions 1 to 8), and then the case of words affected by three or more faults (proposition 9). We start with an illustration of the approach for state faults [29]. Then, we propose algorithms for more complex faults.

Let us consider a simple march test algorithm detecting state faults: {⇕(W0); ⇕(R0); ⇕ (W1) ; ⇕ (R1)}. Here W0 means that a write is performed using the all 0's vector as data, and R0 means that a read is performed with expected value the all 0's vector. In this algorithm a state fault SF < 0/1/- > affecting a cell of any memory word is detected in the 2nd march element, while SF < 1/0/- > affecting a cell of any memory word is detected in the 4th march element. If two SF < 0/1/- > faults affect the same memory word both are detected by a single read (i.e. when this word is read in the 2nd march element). Similarly, if two SF < 1/0/- > faults affect the same memory word both are detected in a single read (i.e. when the word is read in the 4th march element). However, if a SF < 0/1/- > and a SF < 1/0/- > affect the same word, then, the first fault is detected in the 2nd march test sequence and the second fault is detected in the 4th march test sequence. But we need to detect in the same read any two state faults affecting any two cells of the same memory word. The solution is to use the algorithm {⇕(WV_i); ⇕(RV_i); ⇕ (W$\overline{V_i}$); ⇕ (R$\overline{V_i}$)}, obtained by replacing in the algorithm described above the all 0's vector by a vector V_i and the all 1's vector by $\overline{V_i}$. Then, we execute this algorithm k times using each time a different vector V_i, i∈{0, 1, ... k}. This set of binary vectors is selected such that, for any two bit-positions, there is a vector in the set supplying 00 in these positions and another vector in the set supplying 01 in these positions. Let us now consider a memory word comprising two faulty cells. In the above-defined set there is a vector V_{00} such that the bit positions corresponding to the faulty cells take the values 00. Let us execute the test algorithm using V_{00} as test data. Then, if the two faults are SF<0/1/-> both of them are detected when the faulty word is read in the 2nd march element of the algorithm; and if the two faults are SF <1/0/->, both are detected when the faulty word is read in the 4th march element of the test algorithm. Furthermore, in the above-defined set there is a vector V_{01} such that the bit positions corresponding to the faulty cells take the values 01. Let us execute the test algorithm using V_{01} as test data. Then, if the one fault is SF<0/1/-> and the other is SF<1/0/->, both are detected when the faulty word is read in the 2nd march element; and if the one fault is SF<1/0/-> and the other is SF<0/1/->, both are detected when the faulty word is read in the 4th march element. Thus, executing the algorithm for the above-defined set of vectors V_i guaranties the *single-read double-fault detection* property for all double state faults

affecting the same word.
Such a set consists of k= $\lceil \log_2 m \rceil$ +1 vectors V_i (m being the size of the memory word) [30], and its creation is simple:

- We generate the set of all 2^k binary numbers of k bits. This set contains 2^{k-1} pairs of complementary binary numbers.
- We eliminate any one of the two numbers of each pair, to obtain 2^{k-1} binary numbers.
- We eliminate 2^{k-1} – m of these numbers to obtain m binary numbers of k= $\lceil \log_2 m \rceil$ +1 bits.
- We create a kxm matrix having each of these numbers as a column. The k rows of this matrix give the set of vectors V_i we are looking for.

As an example, for m = 8 the number of vectors V_i is k = 4. The rows of the 4x8 matrix given bellow and created as described above, provide a set of 4 vectors V_i having the above described property.

0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1

The aim of the sections 2.1, 2.2, and 2.3, is twofold: present a methodology enabling achieving the single-read double-fault detection property for faults more complex than state faults in order to pave the way for further developments in this domain; propose test algorithms achieving this property for a comprehensive set of faults.

2.1 FUNCTIONAL FAULT MODELS AND TEST CONDITIONS

Unlinked faults (also known as simple faults) are considered, since the treatment of linked faults is still an open question in the memory test literature. Indeed, as noted in [28] "*the fault space for linked faults as well as the required tests remain still to be worked out*". First, we consider single-cell unlinked faults. As concerning multi-cell faults, similarly to [28] in this work we restrict our analysis to two-cell faults, because they are considered to be the most important class of multi-cell faults. Furthermore, we restrict our analysis to static faults. However, the approach developed hereafter can be used to extend the analysis to dynamic faults too. A systematic classification of all static unlinked functional fault models (FFMs) involving one memory cell (single-cell FFMs) and two memory cells (two-cell FFMs) is presented in [29]. These FFMs are reported in tables 1 and 2. For compactness purposes, in these tables we replaced: the value of the aggressor cell (whether it is 0 or 1) by *a*; the value of the victim cell (whether it is 0 or 1) by *v*; the symbol of the transition of the victim cell (whether it is ↑ or ↓) by ◊.

Table 1. List of single-cell FFMs

#	FFM	Fault Primitives	#	FFM	Fault Primitives
1	SF	$< v/\bar{v}/->$	4	RDF	$<rv/◊/\bar{v}>$
2	TF	$<\bar{v}wv/\bar{v}/->$	5	DRDF	$<rv/◊/v>$
3	WDF	$<vwv/◊/->$	6	IRF	$rv/v/\bar{v}>$

Table 2. List of two-cell FFMs

#	FFM	Fault Primitive	#	FFM	Fault Primitive
1	CFst	$<a; v/\bar{v}/->$	3	CFtr	$<a; vw\bar{v}/v/->$
2	CFds:		4	CFwd	$<a; vwv/\Diamond/->$
2.1	CFds(ra)	$<ra; v/\Diamond/->$	5	CFrd	$<a; rv/\Diamond/\bar{v}>$
2.2	CFds(aw$\bar{a}$)	$<aw\bar{a}; v/\Diamond/->$	6	CFdrd	$<a; rv/\Diamond/v>$
2.3	CFds(awa)	$<awa; v/\Diamond/->$	7	CFir	$<a; rv/v/\bar{v}>$

Memory test algorithms covering various combinations of the above faults are presented in [28]. The detection of all of them is achieved at 22n complexity (March SS algorithm [28]). Optimal tests for these FFMs are given in [31], requiring 19n complexity. In the following, we address test algorithms satisfying the *single-read double-fault detection* (SRDF) property for these FFMs. That is: if any combination of the above faults affects two or more cells of the same memory word, then, there is always a read operation in the test algorithm that detects at least two of them (*single-read double-fault detection* property).

The theoretical challenge in developing test algorithms satisfying the SRDF property is far more complex in comparison with the development of conventional test algorithms. Indeed, detecting in a single read two FFMs affecting two cells of the same memory word is equivalent to testing a duplex FFM consisting in the combination of these two FFMs. Thus, while for the fault models of tables 1 and 2 a conventional test algorithm will have to test 6 single-cell and 9 two-cell FFMs (considering read CFds, transition CFds, and non-transition CFds as 3 cases), the SRDF test algorithm will have to test 225 duplex FFMs (all possible combinations of two of the above FFMs). The increase of fault cases is ever sharper if we consider the number of sensitizing states involved in each double fault (e.g. from 4 states related to the values of the victim and aggressor cells in a single two-cell FFM to 16 states for a duplex FFM composed of two two-cell FFMs). The increase of the number of faults is yet sharper as, in a memory word comprising 39 cells (32 data bits plus 7 ECC bits) each FFM can affect any of the 39 cells, while for the same word size each duplex FFM can affect any of the 741 pairs of cells. Furthermore, as the two FFMs composing a duplex FFM affect the same memory word, sensitizing the one FFM may desensitize the other FFM, making even more complex the simultaneous detection of both FFMs. For instance, a write of the faulty memory word sensitizing the one FFM will destroy the sensitization of the other FFM. To cope with these complexities we developed a formal framework comprising numerous lemmas (16) and propositions (8).

We start with a trivial lemma listing a set of conditions that guaranty detecting the set of all single-cell FFMS (SF, TF, WDF, RDF, DRDF, IRF), and the two-cell FFMs CFst and CFds.

Lemma 1: A test algorithm that satisfies each of the conditions i to x given bellow, guaranties detecting each of the single-cell faults (SF, TF, WDF, RDF, DRDF, IRF) and the two-cell faults of types CFst and CFds, by some of its read operations performed over the victim cell (referred hereafter as *victim-detection read* or more simply as *detection read*):

i. The victim cell is at state v during the *victim-detection read* (***SF*** faults related condition).
ii. The operations vwv (for ***WDF*** faults) or rv (for ***DRDF*** faults) is performed over the victim cell before the *victim-detection read*, and no write is performed over the victim cell between these operations.
iii. The *victim-detection read* is performed when the victim is at state v (***RDF*** and ***IRF*** faults).
iv. The operations $\bar{v}wv$ is performed over the victim cell before the *victim-detection read* and no write is performed over the victim cell between these operations (***TF*** faults).
v. There is an instant of the test algorithm preceding the instant of the *victim-detection read* such that: the aggressor cell is at state a; the victim cell is at state v; and no write is performed over the victim between these two instants (***CFst*** fault).

vi. The operation r*a* is performed over the aggressor cell at some instant preceding the *victim-detection read* during which the victim cell has a particular value *v*, and no write is performed over the victim cell between these instants (***CFds(ra)*** faults).

vii. Same as vi with r*a* replaced by ***awa (CFds(awa)*** faults).

viii. Same as vi with r*a* replaced by $aw\bar{a}$ (***CFds(aw$\bar{a}$)*** faults).

ix. Conditions v, vi, vii, and viii are realized for both *a*=0 and *a*=1.

x. The test is executed several times using test data that supply to the victim cell both values *v*=0 *a*nd *v*=1 during the realization of each of the above conditions.

Lemma 1 can be proven trivially.

As mentioned earlier, the analysis and derivation of test algorithms achieving the *single-read double-fault detection* property is far more complex than for conventional tests. Thanks to the unlinked property of the fault models, some simplification of this task can be done by means of a lemma described next. Let us consider a memory word W in which k *distinct* cells (to be referred as victim cell) are affected by faults, and let $k \geq 2$. A victim cell can be affected by multiple FFMs. Thus, a number of FFMs larger than k can affect the k victim cells of W. Let W_f be the set of FFMs affecting the word W.

Lemma 2: If a test algorithm TA detects in the same read the faults of any subset W_f' of the set of faults W_f affecting a memory word W, such that the faults of W_f' affect at least two victim cells of W, then, TA achieves the *single-read double-fault detection* property when word W is affected by the set of faults W_f.

Proof: Since the faults are unlinked, the detection of a fault is not masked by the presence of other faults. Thus, if TA detects in a single read the faults of W_f', it will also detect these faults in the presence of all faults of W_f. As there are at least two victim cells in W_f', this read will detect at least two errors, identifying W as a word affected by at least two faults. **QED**

Thanks to this lemma, we only need to treat double faults [f1, f2], such that f1 and f2 affect two distinct cells of the same memory word, and each of them is a FFM of the table 1 or 2. Thus, all our proofs will demonstrate the *single-read double-fault detection* property for double faults only.

2.2 SRDF TEST ALGORITHMS

Based on lemma 1 we propose the March SRDF1 algorithm (shown in figure 1) that satisfies the *single-read double-fault detection* property for all single-cell FFMs (SF, TF, WDF, RDF, DRDF, IRF), and two important classes of two-cell FFMs (CFst and CFds). March SRDF1 comprises three march elements (M_0, M_1, M_2). M0 has one operation. M1 has six operations (M_{11} through to M_{16}). M2 has seven operations (M_{21} through to M_{27}).

M0 $\{\Updownarrow(wV_i);$

M1 $\Uparrow(\underset{M_{11}}{rV_i}, \underset{M_{12}}{w\bar{V}_i}, \underset{M_{13}}{w\bar{V}_i}, \underset{M_{14}}{r\bar{V}_i}, \underset{M_{15}}{wV_i}, \underset{M_{16}}{wV_i});$

M2 $\Uparrow(\underset{M_{21}}{rV_i}, \underset{M_{22}}{rV_i}, \underset{M_{23}}{w\bar{V}_i}, \underset{M_{24}}{w\bar{V}_i}, \underset{M_{25}}{r\bar{V}_i}, \underset{M_{26}}{wV_i}, \underset{M_{27}}{wV_i})\};$

Figure 1. March SRDF1

Let m be the number of bits of a memory word. A march element uses the same m-bit binary vector V_i (in direct V_i and complementary $\bar{V}_i$ form) as test data in all memory addresses. We can execute a march test several times using each time a different binary vector V_i. A set of m-bit binary vectors V_i is a *two-covering set* of m-bit binary vectors if: for any pair of bit positions each of the values 00, 01, 10, and 11 - appears in some vector Vi of the set. Using such a set is useful as, for a pair of faulty cells belonging to a

memory word, it can allow applying on the victim and/or aggressor cells the value combinations required for satisfying the *single-read double-fault detection* property. Attention has to be paid to the case where some aggressor and victim cells occupy the same bit positions in their respective memory words. This is because in this case the vectors V_i will supply the identical values to these cells and this could invalidate the test. The possible coincidences of the bit positions of the victim and aggressor cells are: the bit position(s) of the aggressor cell(s) coincide(s) with the bit position(s) of the victim cell(s) and/or the two aggressor cells are in the same bit position (including the case where the aggressor cells of the two faults are identical - same bit position and same word).

Please note that, the case where the bit positions of the two victim cells coincide is not an issue. This is because such a double fault can only produce single errors in the faulty memory word, while we are tracking double faults that could produce double-errors in the same memory word.

Proposition 1: The test produced by executing March SRDF1 for each vector V_i of a two-covering set satisfies the *single-read double-fault detection* property for any double fault [f1, f2] such that f1 and f2 belong to the set of faults comprising the single-cell FFMs and the CFst and CFds two-cell FFMs.

Proof: Our challenge is to *detect in the same read* any pair of FFMs affecting two cells of the same memory word. We designed March SRDF1 in a manner that this is accomplished in the M_{22} read operation. Table 3 presents the sensitizing operation/condition that enables detecting in M_{22} each FFM considered in the statement of the proposition. The first column of table 3 lists these FFMs. The second column specifies the relation between the address @a of the aggressor and the address @v of the victim. The third column shows the sensitizing operation/condition enabling detecting the fault in M_{22}. Note that, the second column is empty for all single-cell FFMs (there is no aggressor cell for such faults).

Table 3. Conditions for detecting FFMs in M_{22}

FFMs	Relation @a/@v	Sensitization
SF $< v/\bar{v}/->$	-	M_{22}
TF $< \bar{v}wv/\bar{v}/->$	-	M_{15}
WDF $<vwv/\Diamond/->$	-	M_{16}
RDF $<rv/\Diamond/\bar{v}>$	-	M_{21}
DRDF $<rv/\Diamond/v>$	-	M_{21}
IRF $rv/v/\bar{v}>$	-	M_{22}
CFst $<a; v/\bar{v}/- >$	@a>@v	M1: a=0 & a=1 $\forall V_i$
CFst $<a; v/\bar{v}/- >$	@a<@v	M2: a=0 & a=1 $\forall V_i$
CFds $<ra; v/\Diamond/->$	@a>@v	M_{11}; M_{14}: r0 & r1 $\forall V_i$
CFds $<ra; v/\Diamond/->$	@a<@v	M_{21}; M_{25}: r0 & r1 $\forall V_i$
CFds $<aw\bar{a}; v/\Diamond/->$	@a>@v	M_{12}; M_{15}: 0w1 & 1w0 $\forall V_i$
CFds $<aw\bar{a}; v/\Diamond/->$	@a<@v	M_{23}; M_{26}: 0w1 & 1w0 $\forall V_i$
CFds $<awa; v/\Diamond/->$	@a>@v	M_{13}; M_{16}: 0w0 & 1w1 $\forall V_i$
CFds $<awa; v/\Diamond/->$	@a<@v	M_{24}; M_{27}: 0w0 & 1w1 $\forall V_i$

For CFst faults: march element M1 writes on the aggressor cell both the 0 and the 1 values regardless to the value of vector V_i. Furthermore, for @a>@v no write operation is performed over the victim cell between these writes and the M_{22} read of the victim cell. Thus, condition v of lemma 1 is met in M1 for @a>@v and for both values a=0 and a=1 of the aggressor cell. For @a<@v, the same holds true for the

writes performed on the aggressor cell in M2. Thus, condition v of lemma 1 is satisfied for both values a=0 and a=1 also for @a<@v. The third column of the table reports these facts for CFst faults. For all other faults, the operations realizing the conditions i, ii, iii, iv, vi, vii, and viii for @a>@v and for @a<@v are also reported in the third column. As reported in this column, *these operations realize the conditions i, ii, iii, iv, vi, vii, viii for both a =0 and a= 1, regardless of the value of vector V_i.* Thus condition ix of lemma 1 is also satisfied. Hence, according to lemma 1, executing SRDF1 for a vector V_i will detect in the read operation M_{22} of a victim word any fault of the type SF, TF, WDF, RDF, DRDF, IRF, CFst, or CFds for the value *v* supplied by V_i to the victim cell of this fault. As the set of vectors V_i is a two-covering set, each of the values 00, 01, 10 and 11 is applied to any pair of bit positions by some vector V_i. Thus, if any two cells of a memory word are affected by any pair of faults of the types SF, TF, WDF, RDF, DRDF, IRF, CFst, or CFds, there is a vector V_i that enables detecting both faults in operation M_{22}, provided that there are no bit positions coincidences. Furthermore, bit-positions coincidences are not an issue because: on the one hand, as justified earlier this issue does not concern the coincidence of the positions of the victim cells, and on the other hand, *all sensitizing operations/states of the aggressor cells are supplied by the test algorithm regardless of the value of vector V_i.* Thus, supplying the same value by vector V_i to coinciding aggressor cells or to coinciding aggressor and victim cells is not an issue. **QED**

Note that, a 2-covering set of vectors V_i can be obtained easily by taking the set of vectors used in the beginning of this chapter (in order to achieve the *single-read double-fault detection* property for state faults), and adding their complements. For instance for m = 8 (8-bit words) a 2-covering set is given in the matrix below (the rows of this matrix are the vectors of the 2-covering set, and are obtained by adding the complements of the 4 rows of the matrix generated the beginning of this chapter.

0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1
1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0
1	1	0	0	1	1	0	0
1	0	1	0	1	0	1	0

The cardinality of the covering set of vectors V_i generated by the above method is $k = 2\lceil \log_2 m \rceil + 2$ (which is the minimal cardinality for a covering set). For 22-bits word size (16 data bits and 6 code bits) we have k = 12. For 39-bits word size (32 data bits and 7 code bits) we have k = 14. For 72-bits word size (64 data bits and 8 code bits) we have k = 16. Thus, to satisfy the *single-read double-fault detection* property for a memory using words of 32 data bits and 7 check bits, we have to execute March SRDF1 for 14 vectors Vi.

March SRDF1 was designed to meet the *single-read double-fault detection property* for SF, TF, WDF, RDF, DRDF, IRF, CFst or CFds faults, but no consideration of the remaining two-cell faults has been taken during its construction. Thus, in fig. 2 and 3, we introduce two slightly more lengthy algorithms (March SRDF2 and March SRDF2'), which are more convenient for covering the remaining FFMs. Before addressing the remaining FFMs, we establish in corollary 1 the validity of March SRDF2 and March SRDF2' for SF, TF, WDF, RDF, DRDF, IRF, CFst or CFds faults. Then, we use them to design test algorithms that also cover the missed two-cell FFMs.

Corollary 1: The test produced when March SRDF2 is executed for each vector V_i of a two-covering- set satisfies the *single-read double-fault detection* property for SF, TF, WDF, RDF, DRDF, IRF, CFst or CFds faults. This is also true for March SRDF2'.

Short proof: Considering M22 as victim-detection read, March SRDF2 is designed similarly to SRDF1 to provide all sensitizing states/operations for the faults SF, TF, WDF, RDF, DRDF, IRF, CFst, and CFds, as

well as, for the sensitizing states/operations of the two-cell faults CFst and CFds, to provide both the a = 0 and the a =1 values of the aggressor cell. Thus, the corollary is proven similarly to proposition 1. This also holds true for SDRF2', which is identical to SDRF2 but uses inverse addressing order. **QED**

$\{\Updownarrow(\ W v_i);$
$\Uparrow(\ R v_i,\ W\overline{V_i}\ ,\ W v_i, W v_i, W\overline{V_i}\ , W\overline{V_i}\ , R\overline{V_i}\ );$
$\quad M_{11}\quad M_{12}\quad\ M_{13}\quad M_{14}\quad M_{15}\quad M_{16}\quad M_{17}$
$\Uparrow(R\overline{V_i}\ , R\overline{V_i}\ , W v_i, R v_i,\ W\overline{V_i}\ ,\ W\overline{V_i}\ , W v_i, W v_i)\};$
$\ M_{21}\ M_{22}\ M_{23}\ M_{24}\ M_{25}\ \ M_{26}\ \ M_{27}\ M_{28}$

Figure 2. March SRDF2

$\{\Updownarrow(\ W v_i);$
$\quad M_0$
$\Downarrow(\ R v_i,\ W\overline{V_i}\ ,\ W v_i, W v_i, W\overline{V_i}\ , W\overline{V_i}\ , R\overline{V_i}\ );$
$\quad M_{11}\quad M_{12}\quad\ M_{13}\quad M_{14}\quad M_{15}\quad M_{16}\quad M_{17}$
$\Downarrow(R\overline{V_i}\ , R\overline{V_i}\ , W v_i, R v_i,\ W\overline{V_i}\ ,\ W\overline{V_i}\ , W v_i, W v_i)\};$
$\ M_{21}\ M_{22}$

Figure 3. March SRDF2'

Since a two-cell FFM involves a victim cell and an aggressor cell, if two two-cell FFMs affect two cells of a memory word, four cells will be involved, requiring all value combinations in four bit positions for detecting all combinations of such FFMs. Applying all value combinations in any 4-bit positions requires using a 4-bit covering set of vectors V_i. Nevertheless in proposition 1 and corollary 1 we were able to cope with two important classes of two-cell FFMs (CFst and CFds) by using a two-covering set of binary vectors V_i. This was possible because *CFst and CFds faults are sensitized when the victim cell is in a particular state*. Thus, we can employ a march element (like element M1 or element M2 in SRDF1), which performs all possible operations (RV_i ; $R\overline{V_i}$; V_iWV_i; $\overline{V_i}\ W\overline{V_i}$; $V_iW\overline{V_i}$; $V_iW\overline{V_i}$). These operations enable the march element to sensitize the fault for both the 0 and the 1 states of each aggressor cell, as well as for all possible operations performed over each aggressor cell (0W0, 0W1, 1W1, 1W0, R0, R1), regardless of the value of V_i. As a matter of fact, we only need to use a set of binary vectors V_i that provides to the two victim cells all state combinations (00, 01, 10, 11), which can be done by using a 2-covering set of binary vectors V_i. However, for the remaining two-cell faults (CFtr, CFwd, CFrd, CFdrd, and CFir), the situation is more difficult as stated bellow.

Remark 1: To sensitize a CFtr, CFwd, CFrd, CFdrd, or CFir fault the aggressor cell should have a particular value at an instant at which the victim cell undergoes a particular read or write operation. Thus, contrary to CFst and CFds, a march element cannot sensitize a CFtr, CFwd, CFrd, CFdrd, or CFir fault for both states of the aggressor cell, since the aggressor cell will have a precise (unique) value when the march element performs the required operation over the victim cell.

Remark 2: For CFtr, CFwd, CFrd, CFdrd, or CFir the sensitizing operations are performed over the victim cells. Thus, a march element cannot perform all sensitizing operations because a sensitizing write can mask the sensitization of a fault produced by a previous operation.

From these remarks, achieving the *single-read double-fault detection* property for the remaining two-cell faults is more difficult, and using a 2-covering sets of vectors V_i may not be sufficient for achieving this

property for all of them. In the following we analyse in more details this issue in order to determine all faults that can be addressed by means of 2-covering sets and propose the related algorithms.

For CFtr, CFwd, CFrd, CFdrd, and CFir faults, the operation vwv, vw$\bar{v}$ or rv performed over the victim cell for sensitizing the fault is referred hereafter as ***victim sensitizing operation,*** and the associated value v as ***victim sensitizing-state***. The value ***a*** that has the aggressor cell during the victim sensitizing operation is referred as ***aggressor sensitizing-state*.**

Lemma 3: The following conditions are sufficient and necessary for detecting the following FFMs: CFtr <a; vw$\bar{v}$/v/->, CFwd <a; vwv/◊/- >, CFrd <a; rv/◊/$\bar{v}$ >, CFdrd <a; rv/◊/v>, CFir <a; rv/v/$\bar{v}$> by a given read operation performed over the victim cell (*victim detection read*):

i.a **Detection Conditions for CFtr <a; vw$\bar{v}$ / v / - >:** the operation vw$\bar{v}$ has to be performed over the victim cell (*victim-sensitizing operation*) before the *victim detection read*; no wv write is performed over the victim cell between these *operations*; and no w$\bar{v}$ write is performed over the victim cell between these operations if, in the meantime, the state of the aggressor has been changed.

i.b **Detection Conditions for CFwd <a; vwv / ◊ / - >:** the operation vwv has to be performed over the victim cell (*victim-sensitizing operation*) before the *victim-detection read*, and no write is performed over the victim cell between these operations.

i.c **Detection Conditions for CFrd < a; rv / ◊ / $\bar{v}$ >**: the *victim-sensitizing operation* can be the *victim-detection read* itself or a read preceding it; and, in the latter case no write operation is performed over the victim cell between these operations.

i.d **Detection Conditions for CFdrd < a; rv / ◊ / v >:** the operation rv has to be performed over the victim cell (*victim-sensitizing operation*) before the *victim-detection read* and no write is performed over the victim cell between these reads.

i.e **Detection Conditions for CFir < a; rv / v / $\bar{v}$ >**: the *victim-sensitizing operation* is (necessarily) the *victim-detection read* itself.

ii. **In all cases**: the state of the aggressor cell during the *victim-sensitizing operation* is equal to a.

Lemma 3 can be proven trivially from the definitions of CFtr, CFwd, CFrd, CFdrd, and CFir faults. Also, we check trivially that if the conditions of lemma 3 are met for all value combinations 00, 01, 10 and 11 of a and v, then all CFtr, CFwd, CFrd, CFdrd, or CFir faults are detected by the *victim-detection read*.

Remark 3: From Lemma 3, any number of reads over the victim cell can be performed between the *victim-sensitizing operation* and the *victim-detection read*.

Remark 4: Generally, a write performed over the victim after a fault-sensitizing operation will destroy the fault sensitization. However, for CFtr < a; vw$\bar{v}$ / v / - >, if the aggressor state is a and the victim state is v, performing the operation w$\bar{v}$ leaves the victim at the state v (which is erroneous). Thus, if the aggressor state is still a and we perform again w$\bar{v}$, it will again leave the victim at the state v. Hence the second w$\bar{v}$ does not desensitize the CFtr fault. Consequently, we can perform the two operations w$\bar{v}$; w$\bar{v}$, in order to both: sensitize the CFtr < a; vw$\bar{v}$ / v / - > with the first w$\bar{v}$, and sensitize CFwd < a; $\bar{v}$w$\bar{v}$ / v / - > with the second w$\bar{v}$, without affecting the sensitization of CFtr by the second w$\bar{v}$.

Based on remarks 1, 2, 3, 4 and lemma 3, let us determine how march test algorithms can sensitize and detect CFtr, CFwd, CFrd, CFdrd, and CFir faults.

Lemma 4:

i. A read operation (*victim-detection read*) can detect a CFtr, CFwd, or CFir fault for only one state of the aggressor cell.

ii. A read operation (*victim-detection read*) can detect a CFrd or CFdrd fault for both 0 and 1 states of the aggressor cell.

Proof of part i: In the case of ***CFwd*** faults, from condition i.b of lemma 3, no write can be performed over the memory word containing the victim cell between the write sensitizing this fault and the read detecting it.

This means that a read operation can detect a CFwd fault sensitized by a single write operation, and thus, for only one state of the aggressor cell (the state that has the aggressor during this write).

From condition i.a of lemma 3, either no write is performed over the memory word containing the victim cell of a ***CFtr*** fault between a $v w\bar{v}$ transition sensitizing this fault and its detection read (hence, as above, the fault is sensitized by a single write and thus for a single value of the aggressor cell), or a $w\bar{v}$ is performed but the state of the aggressor cell has not been changed in the mean time (hence the fault is sensitized by several writes but each time the aggressor cell has the same state). In both cases the read can detect a CFtr fault for only one state of the aggressor.

From condition i.e of lemma 3, the sensitizing read of a ***CFir*** fault coincides with its detection read. Thus a read detects a CFir fault for only one state of the aggressor cell (i.e. for the state that has the aggressor cell during this read). **QED.**

Proof of part ii: From i.c and i.d, it is not forbidden to perform a read over the victim cell between a first read sensitizing a CFrd or a CFdrd faults and its detection read. Thus, after a first read sensitizing the fault and performed when the aggressor cell has a value *a*, we can perform a write over the aggressor cell to invert its state to $\bar{a}$ and then perform a second read to sensitize the CFrd or the CFdrd fault for the state $\bar{a}$ of the aggressor cell. A third read will be able to detect each of these faults (e.g. the one having *a* as *aggressor-sensitizing state* and the one having $\bar{a}$ as *aggressor-sensitizing state*). Note that the faults detected by this read have the same *victim-sensitizing state*, since no write has been performed over the victim cell between the first and the third read. Note also that for CFrd faults the second and third reads used in the above analysis can coincide. **QED.**

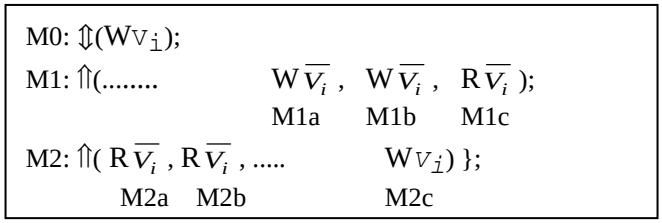


Figure 4: March test algorithm structure developed from lemmas 1 and 4.

Based on lemmas 3 and 4 and their proofs, we derive the march test algorithm structure shown in figure 4. For such algorithms we can show the following lemma.

Lemma 5. Let us consider march test algorithms comprising at least 2 march elements M1 and M2 having the following characteristics: at the beginning of M1 the state of all memory words is V_i; the last three operations of M1 (referred as M1a, M1b, M1c) are W$\overline{V_i}$, W$\overline{V_i}$, R$\overline{V_i}$; the first two operations of M2 (referred as M2a, M2b) are R$\overline{V_i}$, R$\overline{V_i}$; and the last operation of M2 (referred as M2c) is WV_i, as shown in figure 4. Then, for such an algorithm the following properties hold true:

i. The state of the aggressor cell is inverted between the instant in which M1 visits the word containing the victim cell, and the instant in which M2 visits the word containing the victim cell, regardless of the relation of the addresses of the victim and the aggressor cells.
ii. CFtr faults are sensitized by M1a and detected in M1c as well as in M2b.[5]
iii. CFwd faults are sensitized by M1b and detected in M1c and M2b.
iv. CFrd faults are sensitized by M1c and detected in M1c and M2b, and also sensitized by M2b and detected in M2b. From point i above, the *aggressor-sensitizing state* in M2b is the inverse of the *aggressor-sensitizing state* in M1c.

[5] The fault is also detected in M2a, but this detection is not exploited hereafter. Sensitizations and detections for other faults not exploited in our analysis are also omitted in the subsequent text.

v. CFdrd faults are sensitized by M1c and detected in M2b and also sensitized by M2a and detected in M2b. As above, the *aggressor-sensitizing state* in M2a is the inverse of the *aggressor-sensitizing state* in M1c.
vi. CFir faults are sensitized by M1c and detected in M1c, and also sensitized by M2b and detected in M2b. The *aggressor-sensitizing state* in M2b is the inverse of the *aggressor-sensitizing state* in M1c.

Proof: Point i of lemma 5 can be proven trivially by observing that M1 inverses that states of the memory cells and so do M2. The rest of lemma 5 can be proven trivially based on lemma 3. **QED**

Lemma 6: The *victim sensitizing state* is the same for all CFtr faults detected by an algorithm obeying the structure described in lemma 5. This is also true for CFwd faults, CFrd faults, CFdrd faults and CFir faults.

Proof: This lemma can be proven trivially since any CFtr fault is sensitized by the same operation (M1a). Thus, we have a unique *victim-sensitizing state* for CFtr faults. The same holds true for CFwd faults, which are sensitized by operation M1b. Operations performed at several positions of the algorithm can sensitize each of the remaining faults (e.g. operations M1c, M2a and M2b for CFir). However all these operations are identical ($R\overline{v_i}$). Thus, all of them involve the same *victim sensitizing state*. **QED**

Lemma 7: Let us consider a double fault [f1, f2] composed of two faults f1 and f2 such that: f1 and f2 are of the types CFtr, CFwd, CFrd, CFdrd, or CFir, and f1 and f2 affect the same memory word (*victim word*). Let us partition these double faults into the 7 categories (i, ii, iii, iv, v, vi, and vii) according to the types of faults f1 and f2, as described in table 4. Let a1 and a2 be the values that have the aggressor cells of f1 and f2 at the instant in which the march element M1 of an algorithm obeying the structure described in lemma 5 visits the victim word. Then, for the fault [f1, f2] of each category i, ii, iii, iv, v, vi, and vii, such that both f1 and f2 are detected in M1c, or in M2a, or in M2b of this algorithm, column 4 of table 4 gives the values of the pairs of the *aggressor-sensitizing states*.

Table 4. Pairs of aggressor sensitizing states in algorithms obeying the structure described in lemma 5

Fault categories	f1	f2	Pairs of Aggressor Sensitizing States
i	CFtr or CFwd	CFrd or CFdrd	(a1, a2), (a1, $\overline{a2}$)
ii	CFtr or CFwd	CFir	(a1, a2), (a1, $\overline{a2}$)
iii	CFir	CFrd	(a1, a2), ($\overline{a1}$, a2), ($\overline{a1}$, $\overline{a2}$)
iv	CFir	CFdrd	($\overline{a1}$, a2), ($\overline{a1}$, $\overline{a2}$)
v	CFrd or CFdrd	CFrd or CFdrd	(a1, a2), (a1, $\overline{a2}$), ($\overline{a1}$, a2), ($\overline{a1}$, $\overline{a2}$)
vi	CFtr or CFwd	CFtr or CFwd	(a1, a2)
vii	CFir	CFir	(a1, a2), ($\overline{a1}$, $\overline{a2}$)

Proof: To prove this lemma we need to validate the values of the aggressor sensitizing pairs reported in column 4 of table 4. This can be done trivially by using, for each FFM in the column 2 and 3 of the table, its sensitizing operations from lemma 5, and considering the value that has the aggressor cell during these operations, as detailed bellow for each of the fault categories i, ii, iii, iv, v, vi:

i. If f1 is a CFtr or a CFwd fault and f2 is a CFrd or a CFdrd fault, then, from points ii, iii, iv, and v of lemma 5, M2b detects f1 and f2 for *aggressor-sensitizing states* (a1, a2) as well as for *aggressor-sensitizing states* (a1, $\overline{a2}$).

ii. If f1 is a CFtr or a CFwd fault and f2 is a CFir fault, then, from points ii, iii, and vi of lemma 5 we find that M1c detects f1 and f2 for *aggressor-sensitizing states* (a1, a2); and M2b detects f1 and f2 for *aggressor-sensitizing states* (a1, $\overline{a2}$).
iii. If f1 is a CFir fault and f2 is a CFrd fault, then from points iv, and vi of lemma 5 we find that M1c detects f1 and f2 for *aggressor-sensitizing states* (a1, a2); and M2b detects f1 and f2 for *aggressor-sensitizing states* ($\overline{a1}$, a2) as well as for *aggressor-sensitizing states* ($\overline{a1}$, $\overline{a2}$).
iv. If f1 is a CFir fault and f2 is a CFdrd fault, then, from points v, and vi of lemma 5 we find that M2b detects f1 and f2 for *aggressor-sensitizing states* ($\overline{a1}$, a2), as well as for *aggressor-sensitizing states* ($\overline{a1}$, $\overline{a2}$).
v. If f1 is a CFrd or a CFdrd fault and f2 is a CFrd or a CFdrd fault, then, from points iv, and v of lemma 5 we find that M2b detects f1 and f2 for *aggressor-sensitizing states* (a1, a2), as well as for *aggressor-sensitizing states* (a1, $\overline{a2}$), ($\overline{a1}$, a2), and ($\overline{a1}$, $\overline{a2}$).
vi. If f1 is a CFtr or a CFwd fault and f2 is a CFtr or a CFwd fault, then, from points ii and iii of lemma 5 each of the reads M1c, M2a, and M2b detects f1 and f2 for *aggressor-sensitizing states* (a1, a2).
vii. If f1 is a CFir fault and f2 is a CFir fault, then, from points vi of lemma 5, M1c detects f1 and f2 for *aggressor-sensitizing states* (a1, a2), and M2a as well as M2b detects f1 and f2 for *aggressor-sensitizing states* ($\overline{a1}$, $\overline{a2}$). **QED**

Lemma 8. Let us consider a test algorithm using the vector V_i as data background (i.e. the read and write data are V_i or $\overline{V_i}$). Then, for a given *victim-sensitizing operation* involving the *aggressor-sensitizing state a* and the *victim-sensitizing state v* and a given *victim-detection read*, there is a bijective relationship between *a* and V_{ip} and between *v* and V_{ij} (where V_{ip} and V_{ij} are the values that vector V_i supplies to the bit positions ***p*** and ***j*** that the aggressor and the victim cells occupy to their respective words). In particular, inverting V_{ip} inverts *a* and inverting V_{ij} inverts *v*.

Proof: This lemma is proven trivially by observing that inverting the state V_{ik} of any bit position ***k*** in vector V_i will invert all the occurrences in the test algorithm of the state of bit position ***k*** of each memory word. **QED**

Proposition 2: If we execute an algorithm obeying the structure described in lemma 5 for a 2-bit covering set of vectors V_i, then, the *single-read double-fault* detection property is satisfied for any double fault of category v.

Proof: From lemma 7, for any pair of faults f1, f2 of category v detected by the same read operation of an algorithm having the structure described in lemma 5, the pairs of *aggressor sensitizing states* take all four possible values 00, 01, 10 and 11 whatever are the values of a1 and a2 supplied by a test vector Vi. This is because, from column 4 in the table of lemma 7, these pairs take the values (*a1*, *a2*), (*a1*, $\overline{a2}$), ($\overline{a1}$, *a2*) and ($\overline{a1}$, $\overline{a2}$). From lemma 6, when executing the algorithm for a vector V_i, the value *v1* of the *victim sensitizing state* of fault f1 is always the same and this is also true for the value *v2* of the *victim sensitizing state* of fault f2. Thus, each of the pairs (*a1*, *a2*), (*a1*, $\overline{a2}$), ($\overline{a1}$, *a2*) and ($\overline{a1}$, $\overline{a2}$) is combined with the pair (*v1*, *v2*). Then, from lemma 8 we find trivially that executing the algorithm for a 2-covering set of vectors Vi (which supplies all four possible value combinations to any two bit-positions), will provide all four possible value combinations to the *victim sensitizing states* of the faults f1 and f2. Thus, we obtain all 16 possible combinations for the *aggressor* and *victim sensitizing-states* for the double faults of category v.

Concerning bit positions coincidence issues, the same arguments as in the proof of proposition 1 are valid. **QED**

From lemma 7 (table 4), for the faults of category v, an algorithm having the structure described in lemma 5 supplies all four pairs (*a1*, *a2*), (*a1*, $\overline{a2}$), ($\overline{a1}$, *a2*), ($\overline{a1}$, $\overline{a2}$) of *aggressor sensitizing-states*. Thus, we can use a set of two-covered vectors to achieve t*he single-read double-fault detection* property.

Unfortunately, this is not the case for the faults of the other categories. To cope with this issue, we extend the test algorithm in the manner described in lemma 9.

Lemma 9: If the march elements M1 and M2 of the algorithms described in lemma 5 are replaced by two march elements M1' and M2', which have the same structure as M1 and M2 but use inverse address order, then, for CFtr, CFwd, CFrd, CFdrd and CFir faults, the values of the *victim-sensitizing states* remain unchanged but the values of the *aggressor-sensitizing states* are inverted.

Proof: The values of the *victim-sensitizing states* remain unchanged because the *victim-detection read*s M1a', M1b', M1c', M2a', M2b', M2c' are identical to the *victim-detection read*s M1a, M1b, M1c, M2a, M2b, M2c, including the read values. The values of the *aggressor-sensitizing states* for CFtr, CFwd, CFrd, CFdrd and CFir faults are inverted because: when we visit any memory word W1 in M1' or M2' to perform sensitizing operations over potential victim cells, all other memory words (hence including any word W2 that contains potential aggressor cells) have values that are inverse of the values they have when we visit W1 in M1 or M2. This fact is established trivially by noting that:

- Due to the use of inverse address order, if a word W2 is visited in march element M1' after another word W1, then, W2 is visited in M1 before W1, and vice versa.
- The value of a word W2 after it is visited in M1' is the inverse of its value before it is visited in M1, and vice versa.
- The above two facts hold also true for M2' and M2. **QED**

Lemma 10: A march test algorithm (as the one of figure 5), comprising: two march elements M1 and M2 having the structure described in Lemma 5, and two march elements M1' and M2' having the similar structure but use inverse address order (as described in lemma 9); provides all the four *aggressor-sensitizing states* (*a1*, *a2*), (*a1*, $\overline{a2}$), ($\overline{a1}$, *a2*) and ($\overline{a1}$, $\overline{a2}$) for the fault categories i, ii, iii and iv, and the two *aggressor sensitizing states* (*a1*, *a2*), ($\overline{a1}$, $\overline{a2}$) for the fault categories vi, and vii. Furthermore, the values *v1*, *v2* of the *victim-sensitizing states* are always the same.

Proof: The first part of this lemma comes trivially by considering the *aggressor-sensitizing* pairs of table 4 (from lemma 7) and also their inverses (from lemma 9). The second part of this lemma is also comes trivially from lemma 9 (the values of the *victim-sensitizing states* remain unchanged) and the fact that, for the algorithm structure described in lemma 5, we have already found in lemma 6 that the values of the *victim sensitizing states v1* and *v2* are always the same for the FFMs CFtr, CFwd, CFrd, CFdrd, and CFir, which compose the faults of categories i, ii, iii and iv. **QED**

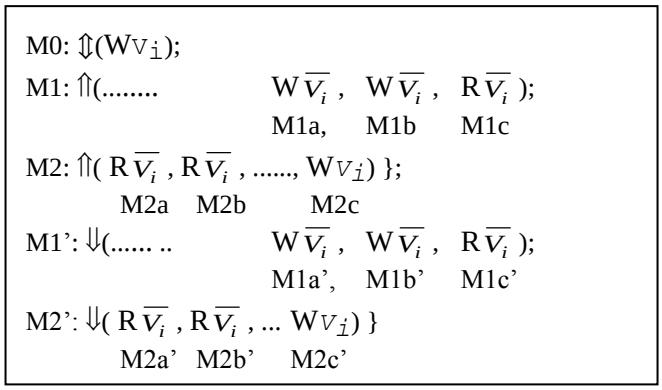


Figure 5. March test algorithm structure related to lemma 10.

Proposition 3: If we execute an algorithm obeying the structure described in lemma 10 for a 2-covering set of vectors V_i, then, the *single-read double-fault* detection property is satisfied for all double faults of categories i, ii, iii, iv and v, and for the half of the faults of categories vi and vii.

Proof: For the faults of category v the proposition has been proven already (see proposition 2).

For the categories i, ii, iii, and iv: based on the results of lemma 10, the proposition is proven similarly to proposition 2 by exploiting the fact that the set of vectors Vi is 2-covering.
For the faults of categories vi and vii, the proof is done similarly by using the fact we use a 2-covering set of vectors V_i, and that from lemma 10 only the half of *aggressor-sensitizing states* ((a1, a2), ($\overline{a1}$, $\overline{a2}$)) are supplied for categories vi and vii.
Concerning bit positions coincidence issues, the same arguments as in the proof of proposition 1 are valid.
QED

Proposition 4: If we execute the algorithm March SRDF3, (shown in figure 6) for a 2-bit covering set of vectors Vi, then, the *single-read double-fault* detection property is satisfied for all double faults composed of single-cell FFMs and two-cell FFMs, except the half of the faults of categories vi and vii.

Proof:

Case 1. Let f1 and f2 be the single faults composing a double fault [f1, f2]. March SRDF3 is composed of March SRDF2 and March SRDF2'. From corollary 1, executing March SRDF2 (and also March SRDF2') for a 2-covering set of vectors Vi, satisfies the *single-read double-fault detection* property for any double fault such that: f1 is a single-cell FFM or a two-cell FFM of the types CFst and CFds and the same is true for f2.

Case 2. We check trivially that March SRDF3 satisfies the structure shown in figure 5. Thus, from proposition 3 the *single-read double-fault* detection property is satisfied for all double faults such that f1 and f2 are two-cell FFMs of the types CFtr, CFwd, CFrd, CFdrd, and CFir, except the half of the double faults of categories vi and vii.

Case 3. To complete the proof we should address the case where f1 is a single-cell FFM or a two-cell FFM of the type CFst or CFds, and f2 is a two-cell FFM of the type CFtr, CFwd, CFrd, CFdrd, or CFir.
When f1 is a two-cell fault of type CFst or CFds, then, from the proof of corollary 1, considering M22 as *detection read*, March SRDF2 provides to the aggressor cell all states/operations required to sensitize such a fault. That is, both the 0 and the 1 states for CFst faults, both the r0 and r1 operations for CFds(r*a*), both the 0w1 and 1w0 for CFds(*a*w$\bar{a}$) and both the 0w0 and 1W1 for CFds(*a*w*a*). Thus, two a*ggressors-sensitizing* states a1 and $\overline{a1}$ are supplied for fault f1. This holds true also for March SRDF2' when M22' is considering as *detection read*. In March SRDF2, the sensitizing operations over the aggressors are performed after the victim is visited in march element M1 and before it is revisited in march element M2. During this period, the value of the victim word is $\overline{V_i}$. Thus, for each fault only one sensitized value is supplied to the victim of f1 (say value v1). We check easily that during the sensitisation of f1 in March SRDF2' the value of the victim word is again $\overline{V_i}$. Thus, for each fault f1, the victim sensitizing value in March SRDF2' is the same as in March SRDF2 (i.e. v1). Therefore, for fault f1, March SRDF2 supplies the following *aggressor-sensitizing* and *victim-sensitizing* values pairs (a1, v1), ($\overline{a1}$, v1), and the same pairs are also supplied in March SRDF2'.
Fault f2 should be detected by the same *detection read* as f1, that is M22 in March SRDF2 or M22' in March SRDF2'. Fault f2 is of the type CFtr, CFwd, CFrd, CFdrd, or CFir. Detecting CFtr, CFwd and CFir faults in M22 of March SRDF2, implies that the sensitizing operation performed over the victim is M15 for CFtr, M16 for CFwd, and M22 for CFir. As there is just one such operation for each of these faults, we have just one *victim-sensitizing* value for each of them. For CFrd and CFdrd faults we can have two sensitizing operations (M17 and M21) but as they are identical they provide a single *victim-sensitizing* value. Thus, we have just one *victim-sensitizing* value for CFrd and CFdrd faults too. Let call v2 the unique *victim-sensitizing* value for any fault of the type CFtr, CFwd, CFrd, CFdrd, or CFir. As operations M15', M16', M17', M21' and M22' are identical to M15, M16, M17, M21, and M22, we will have the same *victim sensitizing* value v2 in March SRDF2 and March SRDF2'. On the other hand, the aggressor sensitizing state for fault f2 takes inverse values in March SRDF2 and March SRDF2' (say a2 and $\overline{a2}$). This gives for f2 the sensitizing pair (a2, v2) in March SRDF2 and ($\overline{a2}$, v2) in March SRDF2'. Thus, the sensitized pairs (a1, v1)

($\overline{a1}$, v1) of f1 are combined in March SRDF2 with the sensitizing pair (a2, v2) of f2, and in March SRDF2' with the sensitizing pair ($\overline{a2}$, v2) of f2. We obtain the following sensitizing quadruples for the double fault [f1, f2]: (a1, v1, a2, v2) ($\overline{a1}$, v1, a2, v2), (a1, v1, $\overline{a2}$, v2) ($\overline{a1}$, v1, $\overline{a2}$, v2). Then, executing March SRDF3 for a 2-covering set of vectors V_i will provide all 4 values to the victim sensitizing states v1 and v2. In this case we find trivially that, the above 4 quadruplets provide all the 16 possible sensitizing quadruples for the double fault [f1, f2].This guaranties that each pair of faults f1 and f2 is detected by the operation M22 or the operation M22' in some of the executions of SRDF3.
Above we considered that f1 is a two-cell fault of the type CFst or CFds. When f1 is a single-cell FFM (i.e. there is no aggressor cell), in the previous arguments we eliminate the aggressor cell of f1 and we obtain the proof for this case too.
Furthermore, as the vectors V_i were used only for applying all possible values to the victim sensitizing states v1 and v2, using similar arguments as in proposition 1 implies that bit positions coincidences do not affect the arguments used in this proof. **QED**

$$
\begin{array}{l}
\{\Updownarrow(\ W V_i); \\
\quad M_0 \\
\Uparrow(\ R V_i,\ \ W\overline{V_i}\ ,\ \ W V_i, W V_i, W\overline{V_i}\ , W\overline{V_i}\ , R\overline{V_i}\ ); \\
\quad M_{11}\ \ M_{12}\ \ \ M_{13}\ \ M_{14}\ \ M_{15}\ \ M_{16}\ \ M_{17} \\
\Uparrow(R\overline{V_i}\ , R\overline{V_i}\ , W V_i, R V_i,\ \ W\overline{V_i}\ ,\ \ W\overline{V_i}\ , W V_i, W V_i)\}; \\
\ M_{21}\ \ M_{22}\ \ M_{23}\ \ M_{24}\ \ M_{25}\ \ \ M_{26}\ \ \ M_{27}\ \ M_{28} \\
\Downarrow(\ R V_i,\ \ W\overline{V_i}\ ,\ \ W V_i, W V_i, W\overline{V_i}\ , W\overline{V_i}\ , R\overline{V_i}\ ); \\
\quad M_{11}'\ \ M_{12}'\ \ \ M_{13}'\ \ M_{14}'\ \ M_{15}'\ \ M_{16}'\ \ M_{17}' \\
\Downarrow(R\overline{V_i}\ , R\overline{V_i}\ , W V_i, R V_i,\ \ W\overline{V_i}\ ,\ \ W\overline{V_i}\ , W V_i, W V_i)\} \\
\ M_{21}'\ \ M_{22}'\ \ M_{23}'\ \ M_{24}'\ \ M_{25}'\ \ \ M_{26}'\ \ \ M_{27}'\ \ M_{28}'
\end{array}
$$

Figure 6. March SRDF3

From tables 1 and 2 we have a total of 15 types of single-cell and two-cell FFMs, which give a total of C_2^{15} + 15 = 120 double-fault combinations. From proposition 4, March SRDF3 achieves the *single-read double-fault detection* property for the large majority of them. Indeed, this is not satisfied only for 4 of these combinations (corresponding to the categories vi and vii of table 4), and only for the half of the faults of these categories. So, the coverage achieved by March SRDF3 could be considered satisfactory. However, if a higher fault coverage is required, we need to cope with the double faults of categories vi and vii. That is, when both faults are of the type CFir; or both faults are of the type CFtr; or both faults are of the type CFwd; or the one fault is CFtr and the other is CFwd. The coverage of these faults is discussed next. First we show that covering these faults requires using a 4-covering set of vectors V_i.
Lemma 11: For any given double fault [f1, f2] let a1, a2, v1, v2 be the values of the bit positions of a vector V_i corresponding to the bit positions of the aggressor cells and victim cells of this fault. A test algorithm that uses the vector V_i and its inverse $\overline{V_i}$ as data in any read and write operation, could achieve the *single-read double-fault detection* property for the double faults of categories vi and vii for the *sensitizing* quadruplets (a1, a2, v1, v2), (a1, a2, $\overline{v1}$, $\overline{v2}$), ($\overline{a1}$, $\overline{a2}$, v1, v2), and ($\overline{a1}$, $\overline{a2}$, $\overline{v1}$, $\overline{v2}$), and *only* for these quadruplets.
Proof: We can easily check that the algorithm shown in figure 7 achieves the *single-read double-fault* detecting property for the double faults of categories vi and vii corresponding to the *sensitizing* quadruplets: (a1, a2, v1, v2), (a1, a2, $\overline{v1}$, $\overline{v2}$), ($\overline{a1}$, $\overline{a2}$, v1, v2), and ($\overline{a1}$, $\overline{a2}$, $\overline{v1}$, $\overline{v2}$). This proves the 1st part of the lemma.

Concerning the 2nd part (only faults corresponding to the quadruplets (a1, a2, v1, v2), (a1, a2, $\overline{v1}, \overline{v2}$), ($\overline{a1}$, $\overline{a2}$, v1, v2), and ($\overline{a1}$, $\overline{a2}$, $\overline{v1}, \overline{v2}$) can be detected), we first note that a test algorithm using V_i and $\overline{V_i}$ as data in its read and write operations can include only the following four operations: WV_i, W$\overline{V_i}$, RV_i and R$\overline{V_i}$. Let i, j, p and q be the bit positions of the aggressor and victim cells of a double fault [f1, f2]. Let a1, a2, v1 and v2, be the values of the bit positions i, j, p and q in vector V_i. Let us consider a test algorithm that includes a read detecting both f1 and f2 (*detection read*).

Let both f1 and f2 be CFtr faults. From the detection conditions of CFtr faults given earlier (lemma 3), we have that: between the operation *v*w$\bar{v}$ sensitizing a CFtr fault and the read operation detecting this fault no w*v* operation can be performed over the victim cell. This implies that if a CFtr fault is sensitized by the operation V_iW$\overline{V_i}$ and is detected by a read, then a CFtr fault sensitized by the operation $\overline{V_i}$WV_i cannot be detected by the same read, and vice-versa. Thus, as f1 and f2 affect the same memory word, if they are detected by the same read they are sensitized by the same operation, which is either V_iW$\overline{V_i}$ or $\overline{V_i}$WV_i. As a consequence, the possible combinations of victim sensitizing states are (v1, v2) and ($\overline{v1}, \overline{v2}$). Also, as the content of a memory word is either vector V_i or its inverse (i.e. $\overline{V_i}$), at each given instant of the test algorithm the values of the aggressor cells are either a1 and a2, or $\overline{a1}$ and $\overline{a2}$. As f1 and f2 are sensitized by the same operation, the *aggressor sensitizing* states will be the values that have the aggressor cells during this operation, yielding to two possible *aggressor-sensitizing* states combinations: (a1, a2), and ($\overline{a1}$, $\overline{a2}$). Thus, the test algorithm could provide only the following combinations of *sensitizing* quadruplets: (a1, a2, v1, v2), (a1, a2, $\overline{v1}, \overline{v2}$), ($\overline{a1}$, $\overline{a2}$, v1, v2), and ($\overline{a1}$, $\overline{a2}$, $\overline{v1}, \overline{v2}$).

The proof is given in similar manner when f1 and f2 are both CFwd faults, as well as when f1 and f2 are both CFir faults.

When the one fault is CFtr and the other is CFwd, the proof can also be done with similar arguments, while paying attention to the fact that the remark 4 related to lemma 3 is not helpful for increasing the victim sensitizing values:

i. The detection condition for CFtr < *a*; *v*w$\bar{v}$ / *v* / - > allows performing any number of w$\bar{v}$ operations between the victim sensitizing operation *v*w$\bar{v}$ and the detection read;
ii. However this is allowed only if the state of the aggressor is the same during the victim sensitizing operation *v*w$\bar{v}$ and the subsequent w$\bar{v}$ operations.

Thus, while point i. seems allowing performing the sensitizing operations of the CFtr and the CFwd faults at different instants of the algorithm, which could be exploited for modifying the values of the aggressor cells between the sensitization operation of f1 and that of f2 (introducing this way the *aggressor sensitizing* pair (a1, $\overline{a2}$)), point ii prevents us from doing it. **QED**

{⇳(WV_i);
⇑(W$\overline{V_i}$, W$\overline{V_i}$, R$\overline{V_i}$);
⇑(WV_i, WV_i, RV_i);
⇓(W$\overline{V_i}$, W$\overline{V_i}$, R$\overline{V_i}$);
⇓(WV_i, WV_i, RV_i)}

Figure 7. Algorithm structure related to lemma 11

Lemma 12: Executing march test algorithms for a 2-covering set or a 3-covering set of vectors V_i do not guaranty satisfying the *single-read double-fault detection* property for all faults of categories vi and vii.

Proof: Let us consider any march test algorithm using a vector V_i as data background. Such an algorithm can perform read and write operations using V_i and $\overline{V_i}$ as data. Thus, the conditions of lemma 11 are satisfied. From this lemma, we have that executing a march test algorithm for a vector Vi can allow us satisfying the *single-read double-fault detection* property for the faults of categories vi and vii corresponding to the sensitizing quadruplets (a1, a2, v1, v2), (a1, a2, $\overline{v1}, \overline{v2}$), ($\overline{a1}$, $\overline{a2}$, v1, v2), and ($\overline{a1}$, $\overline{a2}$, $\overline{v1}, \overline{v2}$) and *only* to these quadruplets. Supplying the variables a1, a2, v1, v2 of these quadruplets with the values of a 2-covering set does not guaranty producing all possible 16 values combinations. This can be shown easily by example. For instance, if we give to the variables a1, a2, v1, v2 of the quadruplets (a1, a2, v1, v2) (a1, a2, $\overline{v1}, \overline{v2}$), ($\overline{a1}$, $\overline{a2}$, v1, v2), and ($\overline{a1}$, $\overline{a2}$, $\overline{v1}, \overline{v2}$) the values of the of the following 2-covering set of 4-bit vectors 0000; 0011; 0101; 1111; 1100; 1010, we obtain the vectors 0000; 0011; 0101; 1111; 1100; 1010, 0110; 1001, which are the half of the 16 possible quadruplet values.
The case concerning the 3-covering sets can also be proven by example. For instance, if we give to the variables a1, a2, v1, v2 of the quadruplets (a1, a2, v1, v2) (a1, a2, $\overline{v1}, \overline{v2}$), ($\overline{a1}$, $\overline{a2}$, v1, v2), and ($\overline{a1}$, $\overline{a2}$, $\overline{v1}, \overline{v2}$) the values of the following 8 quadruplet values 0000; 0011; 0101; 0110; 1001; 1010; 1100; 1111, which form a 3-covering set, we obtain the same set of 8 quadruplet values, which are the half of the 16 possible quadruplet values.
Thus, executing march tests for a 2-covering or a 3-covering set of vectors does not guaranty supplying all the 16 sensitizing quadruplets for the faults of categories vi and vii. **QED**

From lemma 12, 2-covering and 3-covering sets do not guaranty providing the necessary values for supplying all the 16 sensitizing quadruplets for the faults of categories vi and vii. We can supply these values by providing on a1, a2, v1, v2 all 16 quadruplet values by means of a 4-covering set of vectors V_i. On the other hand, this is also possible by providing only 4 selected quadruplet values on a1, a2, v1, v2. For instance, if we provide on a1, a2, v1, v2 the 4 quadruplet values 0000, 0001, 0100, and 0101, the quadruplets (a1, a2, v1, v2) (a1, a2, $\overline{v1}, \overline{v2}$), ($\overline{a1}$, $\overline{a2}$, v1, v2), and ($\overline{a1}$, $\overline{a2}$, $\overline{v1}, \overline{v2}$) give all the 16 possible quadruplet values. We obtain the same result for any four quadruplet values obtained by inverting the two first bits, or the two last bits, or all four bits in any of the four quadruplet values 0000, 0001, 0100, 0101. This gives 256 sets of four quadruplet values, each of whom gives the desired result. Then, we need a set of vectors V_i that provides at each four bits the values of one of these 256 sets of 4 quadruples. This can reduce the number of vectors V_i with respect to a 4-covering set. However, as there are no such sets of vectors in the literature, their creation requires intensive simulated annealing computations, as those done for creating 4-covering sets [32][33], which have to be done. Thus, in this work we use a 4-covering set of vectors V_i, which is lengthy. So, for moderating the test length, we developed a short test algorithm that can be combined with the 4-covering set.
Proposition 5: Executing March SRDF4 (shown in figure 8) for a 4-covering set of vectors V_i satisfies the *single read double-fault detection* property for the double faults [f1, f2] of categories vi and vii, provided that there is no coincidence of the bit position of an aggressor cell with the bit positions of the other aggressor cell or with the bit position of a victim cell.
Proof:
Faults of category vi. In March SRDF4, operation M11 sensitizes the CFtr fault < *a*; *v*w$\bar{v}$ / *v* / - >, where *v* is the value supplied by vector V_i to the bit position of the victim cell; *a* is the value supplied by V_i to the bit position of the aggressor cell when @a > @v, or its inverse when @a < @v (where @a is the address of the aggressor cell and @v the address of the victim cell). Then, operation M13 detects the CFtr fault for these values of *v* and *a*.

M12 sensitizes the CFwd fault < *a*; $\bar{v}w\bar{v}$ / *v* / - >, where *v* is the value supplied by V_i to the bit position of the victim cell; *a* is the value supplied by V_i to the bit position of the aggressor cell when @a > @v, or its inverse when @a < @v. Then, operation M13 detects the sensitized CFwd fault for these values of *v* and *a*.
From the above, if f1 and f2 are CFtr or CFwd faults (double-fault of category vi), M13 will detect the double fault corresponding to the sensitizing quadruplet (a1, a2, v1, v2) if @a > @v, or ($\overline{a1}$, $\overline{a2}$, v1, v2) if @a < @v, determined by four different bit positions of V_i (since according to the statement of the proposition, there are no bit position coincidences for the aggressor and the victim cells). Thus, executing March SRDF4 for each vector V_i of a 4-covering set will produce all 16 value combinations to both the quadruplets (a1, a2, v1, v2) and ($\overline{a1}$, $\overline{a2}$, v1, v2). Thus, each of the 16 possible double faults of category vi will be detected by operation M13 in at least one of these executions.
Faults of category vii. M13 sensitizes and detects the CFir fault. If f1 and f2 are CFir faults (double-fault of category vii), we find similarly a quadruplet (a1, a2, v1, v2) determined by four different bit positions of vector V_i. Thus, executing March SRDF4 for each vector V_i of a 4-covering set, guaranties that each of the 16 possible double faults of category vii will be detected by operation M13 in at least one of these executions. **QED**

$\{\Updownarrow(W V_i);$
$\Uparrow(W\overline{V_i}, W\overline{V_i}, R\overline{V_i});$
M11 M12 M13

Figure 8: March SRDF4

In the following, we discuss the case of faults not covered by proposition 5 due to bit positions coincidences.
Proposition 6: Executing March SRDF5 (shown in figure 9) for a 3-covering set of vectors V_i achieves the *single-read double-fault detection* property for double faults of categories vi and vii for which the bit position of the one aggressor cell coincides with the bit position of the one victim cell and the bit position of the other aggressor cell is different from the bit position of both victim cells.
Proof: We find easily that the March SRDF5 shown in figure 9 achieves the *single-read double-fault detection* property for the double faults of categories vi and vii corresponding to *sensitizing quadruplets* (a1, a2, v1, v2), and (a1, a2, $\overline{v1}$, $\overline{v2}$). If the bit position of the aggressor of f2 coincides with the bit position of the victim of f2, then, the above *sensitizing quadruplets* become (a1, a2, v1, a2), and (a1, a2, $\overline{v1}$, $\overline{a2}$), where a1, and v1 are the values supplied by vector V_i on the bit positions of the of the aggressor and victim cells of fault f1, and a2 is the value supplied by V_i on the common bit position of the aggressor and victim cells of fault f2. Using a 3-covering set of vectors V_i will produce the set of all 8 possible value combinations on (a1, a2, v1), as well as all 8 possible value combinations on (a1, a2, $\overline{v1}$). Then we can check trivially that the values supplied on (a1, a2, v1, a2), and (a1, a2, $\overline{v1}$, $\overline{a2}$) cover all 16 possible quadruplet values.
Obviously, the similar arguments hold when the bit position of the aggressor of f2 coincides with the bit position of the victim of f1, as well as when the bit position of the aggressor of f1 coincides with the bit position of the victim of f1 or with the bit position of the victim of f2. **QED**

$\{\Updownarrow(W V_i);$
$\Uparrow(W\overline{V_i}, W\overline{V_i}, R\overline{V_i});$
$\Downarrow(W V_i, W V_i, R V_i)\}$

Figure 9: March SRDF5

A particular case of faults of categories vi and vii is the case where the aggressor cell of f1 coincides with the aggressor cell of f2. For some of these faults no test algorithm can achieve the *single-read double-fault detection* property. Surprisingly, as shown in the next two lemmas, march test algorithms allow coping with these faults in the context of the present study.

Lemma 13: Executing March SRDF4 for a 3-covering set of vectors V_i and March SRDF5 for a 2-covering set of vectors V_i guaranties the *single-read double-fault detection* property for the double faults [f1, f2] of categories vi and vii in which f1 and f2 share the same aggressor cell (common aggressor) and in which the *aggressor sensitizing value* of f1 is equal to the *aggressor sensitizing value* of f2.

Proof: Let us first consider the case where the two aggressor cells have the same bit position but this bit position is different from the bit position of each of the victim cells. Then, March SRDF4 supplies the *sensitizing quadruplets* (a1, a1, v1, v2). Executing it with a 3-covering set of vectors V_i supplies all 8 quadruplets in which the sensitizing values of the two aggressors are equal.

Let us also consider that the common bit position of the aggressor cells coincides with the bit position of one of the victim cells (e.g. with the bit position of the victim of f1), then, we will have a1 = a2 = v1 and March SRDF5 supplies the *sensitizing quadruplets* (a1, a1, a1, v2). It will also supply (a1, a1, $\overline{a1}$, $\overline{v2}$) as we have seen in the proof of proposition 6. We check trivially that executing March SRDF5 for a 2-covering set of vectors V_i supplies all 8 quadruplets in which the sensitizing values of the two aggressors are equal. **QED**

Lemma 14: No test algorithm can ensure the *single-read double-fault detection property* for the faults of categories vi and vii in which f1 and f2 share the same aggressor cell (common aggressor) and in which the aggressor sensitizing value of f1 is the inverse of the aggressor sensitizing value of f2, but these faults are meaningless in the context of the present study because they never produce double errors.

Proof: Let us first consider the case of a fault of category vi (i.e. f1 is CFtr or CFwd and f2 is CFtr or CFwd). Because f1 and f2 are sensitized by different states of their common aggressor cell, then, it is not possible to sensitize both of them by the same write performed over the word comprising the two victim cells. Then, let us suppose that during system operation or during test, one of the faults is sensitized. To sensitize the second fault another write has to be performed with the common aggressor cell being at a different state. As for this state the first fault is not sensitized, the new value written by the second write over the victim cell of this fault will bring this cell in a correct state. Thus, at any time only one cell can have erroneous value, implying that double errors cannot be produced during test, nor during system operation.

Let us now consider the case where the fault is of category vii (i.e. f1 and f2 are both CFir). Because f1 and f2 are sensitized by different states of the common aggressor cell, then, it is not possible to sensitize both of them by the same write performed over the word comprising the two victim cells. Furthermore, each time one of these faults is sensitized, the victim cell provides an erroneous read value (single-error) but its state remains correct. Then, any subsequent read could again sensitize only one of the two faults, producing again a single-error. It results that the two victim cells can never provide erroneous values in the same read.

Consequently, in both cases (fault of category vi sensitized by opposite states of the common aggressor cell, or fault of category vii sensitized by opposite states of the common aggressor cell), the occurrence of a double error is impossible during test as well as during system operation. Thus, on the one hand there is no test algorithm able to achieve the *single-read double-fault detection* property for these faults, but on the other hand, we do not need to repair the word containing such a fault because it never produces double errors. Therefore, we do not need to achieve the *single-read double-fault detection* property for these faults. **QED.**

Proposition 7 provides the tests enabling covering all faults of categories vi and vii treated so far and few other cases.

Proposition 7: Executing March SRDF4 for a 4-covering set of vectors V_i and March SRDF5 for a 3-covering set of vectors V_i, achieves the *single-read double-fault* detection property for all faults of categories vi and vii excepting half of the faults of the following two cases:

- **a.** The bit positions of the two aggressor cells coincide (but the aggressor cells themselves do not coincide).
- **b.** The bit position of the one aggressor cell coincides with the bit position of one victim cell and the bit position of the other aggressor cell coincides with the bit position of the other victim cell.

Proof: From proposition 5, we cover all faults of categories vi and vii for which there is no coincidence of the bit position of the one aggressor cell with the bit positions of the other aggressor cell or with the bit position of a victim cell. From proposition 6 we cover the cases where the bit position of the one aggressor cell coincides with the bit position of the one victim cell and the bit position of the other aggressor cell is different from the bit position of both victim cells.

From lemmas 13 and 14 we cover the case where the two faults share the same aggressor cell.

Remember that the case where the two victim cells coincide is not of interest in this study (because in this case no double error is produced as the errors occur in the same cell). Note also that if the victim cell of a fault coincides with the aggressor cell of another fault, then, the sensitization of the first fault modifies the state of its victim cell, which is also the state of the aggressor of the second fault. Thus, the sensitization of the first fault can mask the second fault, meaning that the faults are linked. Therefore, as the FFMs of tables 1 and 2 are unlinked, we do not need to consider the case where the aggressor cell of a fault coincides with the victim cell of another fault.

Thus, the only cases of faults of categories vi and vii not yet treated are the following:

- **a.** The bit positions of the two aggressor cells coincide (but the aggressor cells themselves do not coincide).
- **b.** The bit position of the one aggressor cell coincides with the bit position of one victim cell, but these cells do not coincide; and the bit position of the other aggressor cell coincides with the bit position of the other victim cell, but these cells do not coincide.

In the proof of proposition 6 we found that March SRDF5 achieves the *single-read double-fault detection* property for the double faults of categories vi and vii corresponding to the *sensitizing quadruplets* (a1, a2, v1, v2), and (a1, a2, $\overline{v1}$, $\overline{v2}$).

Case a: in this case the above quadruplets become (a1, a1, v1, v2), and (a1, a1, $\overline{v1}$, $\overline{v2}$). As March SRDF5 is executed for a 3-covering set of vectors V_i, all 8 values combinations are supplied on a1, v1, and v2. Then, (a1, a1, v1, v2) gives 8 *sensitizing quadruplets*, and (a1, a1, $\overline{v1}$, $\overline{v2}$) gives the same quadruplets (all 8 quadruplets in which the sensitizing values of the two aggressor cells are equal). As all possible sensitized quadruplets are 16, the *single-read double-fault detection* property is satisfied for the half of the faults of case a).

In **case b** we have two possible situations:

- **b.i.** a1 = v1 and a2 = v2; giving the sensitizing quadruplets (a1, a2, a1, a2), and (a1, a2, $\overline{a1}$, $\overline{a2}$).
- **b.ii.** a1 = v2 and a2 = v1; giving the sensitizing quadruplets (a1, a2, a2, a1), and (a1, a2, $\overline{a2}$, $\overline{a1}$).

Case b.i: as March SRDF5 is executed for a 3-covering set of vectors V_i, all 4 value combinations are supplied on a1 and a2. Then, (a1, a2, a1, a2), and (a1, a2, $\overline{a1}$, $\overline{a2}$) give 8 sensitizing quadruplets: all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the sensitizing states of the victims of f1 and f2; and all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the inverse of the sensitizing states of the victims of f1 and f2.

Case b.ii: As all 4 value combinations are supplied on a1 and a2 we obtain 8 sensitizing quadruplets: all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the

sensitizing states of the victims of f2 and f1; and all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the inverse of the sensitizing states of the victims of f2 and f1.

Thus, the *single-read double-fault detection* property is also satisfied for the half of the faults in the cases b.i and b.ii.

QED

From propositions 4 and 7, we find that executing March SRDF3 for a 2-bit covering set of vectors V_i; March SRDF4 for a 4-covering set of vectors V_i; and March SRDF5 for a 3-covering set of vectors V_i achieves the *single-read double-fault* detection property for all double-faults composed of single-cell FFMS and/or double-cell FFMS (tables 1 and 2), except a very small subset of the faults of categories vi and vii specified in proposition 7 and its proof. Lemma 15 shows that these faults cannot be covered by march test algorithms and will be referred as *march-SRDF-deceptive* faults.

Lemma 15. For the faults of categories vi and vii that are not covered in proposition 7, it is impossible to satisfy the *single-read double-fault detection* property by means of march test algorithms.

Proof. The read and write operations in any march test algorithm use V_i and $\overline{V_i}$ as data (where V_i is the vector used as data background). Thus, the conditions of lemma 11 are satisfied. Hence, based on this lemma, a march test algorithm can achieve the *single-read double-fault detection* property for the double faults of categories vi and vii corresponding to the *sensitizing quadruplets* (a1, a2, v1, v2), (a1, a2, $\overline{v1}, \overline{v2}$), ($\overline{a1}$, $\overline{a2}$, v1, v2), and ($\overline{a1}$, $\overline{a2}$, $\overline{v1}, \overline{v2}$), and **only** to these quadruplets (where a1, a2, v1, v2 are the values that vector V_i supplies to the bit positions of the two aggressor and the two victim cells). Thus, for quadruplets different from the above (the "**only**" part of lemma 11), no march test algorithm can satisfy the *single-read double-fault detection* property.

For the faults of case **a.** of proposition 7, the bit positions of the two aggressor cells coincide. Thus, the above quadruplets become (a1, a1, v1, v2), (a1, a1, $\overline{v1}, \overline{v2}$), ($\overline{a1}$, $\overline{a1}$, v1, v2), and ($\overline{a1}$, $\overline{a1}$, $\overline{v1}, \overline{v2}$). If we assume a set of vectors V_i supplying all possible 8 value combinations on the variables a1, v1 and v2, we check easily that each of these quadruplets gives the same 8 value combinations (all the 8 quadruplets in which the sensitizing values of the two aggressor cells are equal). These quadruplets are the same as those covered in case **a.** of the proof of proposition 7. As the *single-read double-fault detection* property cannot be satisfied for the remaining 8 quadruplets (the "**only**" part of lemma 11), proposition 7 is shown for the faults of case **a.**

For the faults of case **b.i** (see proof of proposition 7), we set a1 = v1 and a2 = v2. In this case the four quadruplets of lemma 11 become (a1, a2, a1, a2), (a1, a2, $\overline{a1}, \overline{a2}$), ($\overline{a1}$, $\overline{a2}$, a1, a2), and ($\overline{a1}$, $\overline{a2}$, $\overline{a1}, \overline{a2}$). Using a set of vectors V_i supplying all possible 4 value combinations on the variables a1 and a2 gives 8 sensitizing quadruplets: all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the sensitizing states of the victims of f1 and f2; and all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the inverse of the sensitizing states of the victims of f1 and f2, which are the same as those covered in case **b.i.** of proposition 7. As the *single-read double-fault detection* property cannot be satisfied for the remaining 8 quadruplets (the "**only**" part of lemma 11), proposition 7 is shown for the faults of case **b.i.**

For the faults of case **b.ii** (see proof of proposition 7), we set a1 = v2 and a2 = v1. In this case four quadruplets of lemma 11 become (a1, a2, a2, a1), (a1, a2, $\overline{a2}$, $\overline{a1}$), ($\overline{a1}$, $\overline{a2}$, a2, a1), and ($\overline{a1}$, $\overline{a2}$, $\overline{a2}$, $\overline{a1}$). Using a set of vectors V_i supplying all possible 4 value combinations on the variables a1 and a2 gives 8 sensitizing quadruplets: all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the sensitizing states of the victims of f2 and f1; and all 4 quadruplets in which the sensitizing states of the aggressors of f1 and f2 are equal respectively to the inverse of the sensitizing states of the victims of f2 and f1, which are the same as those covered in case **b.ii.** of proposition 7. As the *single-*

read double-fault detection property cannot be satisfied for the remaining 8 quadruplets (the "**only**" part of lemma 11), proposition 7 is shown for the faults of case **b.ii. QED**

From Lemma 15, the fault cases that are not covered by the proposed march test algorithms cannot be covered by any march tests. The case of these faults is further discussed in section 2.4.

2.3 FAULTS OF MULTIPLICITY HIGHER THAN 2

Lemma 16: Executing March SRDF3 for a 2-bit covering set of vectors V_i, guaranties achieving the *single-read double-fault* detection property for all faults of multiplicity higher than 2, which are not composed exclusively of FFMs of the types CFtr and CFwd nor exclusively of FFMs of the type CFir.
Proof: If a multiple fault is not composed exclusively of FFMs of the types CFtr and CFwd, or exclusively of FFMs of the type CFir, then, there are at least two single faults fi and fj belonging to the single faults composing the multiple fault, such that the double fault [fi, fj] is neither of category vi nor of category vii. Thus, proposition 4 guaranties the detection of [fi, fj] and lemma 2 the detection of the multiple fault. **QED**

As we have shown that we cannot achieve the *single-read double-fault* detection property for all the faults of multiplicity 2 (lemma 15) by means of march test algorithms, one can consider that this is also the case for faults of multiplicity higher than 2. Surprisingly, the following proposition shows that the march test algorithms proposed in the previous sections cover all faults of multiplicity higher than 2. The reason for this outcome is that faults of multiplicity higher than 2 comprise several double faults, and, as we show in the following proposition, it is not possible for all of them to satisfy the conditions that classify them as of *march-SRDF-deceptive* faults. Due to lemma 2, this implies that all the faults of multiplicity higher than 2 are covered by march tests, and in particular by our march tests that cover all double faults except the *march-SRDF-deceptive* ones.
Proposition 8: Executing March SRDF3 for a 2-bit covering set of vectors V_i, March SRDF4 for a 4-covering set of vectors V_i, and March SRDF5 for a 3-covering set of vectors V_i, achieves the *single-read double-fault* detection property for all faults of multiplicity higher than 2.
Proof: We will represent the bit positions of the aggressor and the victim cell of a fault fi as ap(fi) and vp(fi).

Let us consider a memory word W in which k cells c1, c2, …, ck ($k \geq 3$) are affected by faults.

Thanks to lemma 2 it is sufficient to prove the proposition for multiple faults [f1, f2, … fk] affecting k distinct cells of the same memory word (with $k \geq 3$), such that each of the faults f1, f2, … fk is a single FFM. Furthermore, thanks to the same lemma, for a fault [f1, f2, … fk] it is sufficient to prove the proposition just for two of the faults f1, f2, … fk. Let fi, fj be such two faults. From the above: fi is a single FFM; fj is a single FFM; fi and fj affect distinct victim cells. As the victim cells of fi, and fj are distinct and belong to the same memory word, they have different bit positions. This condition can be written as: $vp(fi) \neq vp(fj)$ **(1)**

Thanks to lemma 16 we only need to prove the proposition 8 for: the multiple faults that are composed exclusively of FFMs of the types CFtr and CFwd (in which case, all the double faults comprised in the multiple fault will belong to the category vi); and the multiple faults that are composed exclusively of FFMs of the type CFir (in which case, all the double faults comprised in the multiple fault will belong to the category vii).
In proposition 7, for a pair of faults [fi, fj] we can:
- write condition **a** as:
  a) ap(fi) = ap(fj),
- split condition **b** in two cases written as:

b1) ap(fi) = vp(fi) and ap(fj) = vp(fj)
b2) ap(fi) = vp(fj) and ap(fj) = vp(fi).

From proposition 7, the *single-read double-fault* detection property is guaranteed to be satisfied for [fi, fj] if [fi, fj] does not satisfy condition **a**, neither condition **b1**, nor condition **b2**. Thus, it is sufficient to show that, for at least one double fault comprised in a multiple fault [f1, f2, … fk], none of these conditions is satisfied.

Let us consider the following conditions concerning the bit positions of the aggressor and victim cells of [f1, f2, … fk]:

c.1) The aggressor cell of each fault has the same bit position as the victim cell of the fault.
c.2) All aggressor cells have the same bit position.

There are three possible cases concerning the validity of these conditions: both c.1 and c.2 are false, which can be represented as **!c.1 and !c.2**; c.1 is true, which can be represented as **c.1**; c.2 is true, which can be represented as **c.2**. Below we discuss each of them.

[f1, f2 … fk] satisfies !c.1 and !c.2: In this case we find trivially that the following two conditions hold:

!c.1: For at least one of the faults f1, f,2 … fk the bit position of its aggressor cell is different from the bit position of its victim cell. Let f1 be such a fault.

!c.2: Since c.2 is false, the bit position of the aggressor cell of f1 is different from the bit position of the aggressor cell of at least one of the faults f2 … fk. Let f2 be this fault.

Thus, conditions !c.1 and !c.2 can be written as:

!c.1 ap(f1) ≠ vp(f1)
!c.2 ap(f1) ≠ ap(f2).

For the double fault [f1, f2], **!c.2)** excludes the condition **a)** of proposition 7 and **!c.1)** excludes condition **b1)**. Thus, in the case **!c.1 and !c.2** *we just need to exclude condition* **b2)**.

There are two possible relationships concerning ap(f1) and vp(f2):

- either ap(f1) ≠ vp(f2) **(2)**
- or ap(f1) = vp(f2) **(3)**

If ap(f1) ≠ vp(f2) condition **b2)** is excluded for [f1, f2]. Thus, proposition 8 is proven for case **(2)**.

Hence, we only need to treat the case **(3)**. This can be combined with two possible cases concerning ap(f2) and vp(f1): ap(f2) ≠ vp(f1) and ap(f2) = vp(f1), giving:

- ap(f1) = vp(f2) and ap(f2) ≠ vp(f1) **(3.1)**
- ap(f1) = vp(f2) and ap(f2) = vp(f1) **(3.2)**

The relation ap(f2) ≠ vp(f1) in **(3.1)** excludes **b2)** for the fault [f1, f2]. Thus, proposition 8 is proven for case **(3.1)**.

Hence, we only need to treat case **(3.2)**. Since we analyze faults of multiplicity higher than 2, we can consider a third fault in the list f1, f2 … fk (e.g. fk). Then, **(3.2)** can be combined with two possible cases concerning ap(f1) and ap(fk): ap(f1) = ap(fk) and ap(f1) ≠ ap(fk), giving::

ap(f1) = vp(f2), ap(f2) = vp(f1), ap(f1) = ap(fk) **(3.2.1)** ap(f1) = vp(f2), ap(f2) = vp(f1), ap(f1) ≠ ap(fk) **(3.2.2)**

In case of **(3.2.1)**, considering the relation ap(f1) ≠ ap(f2) from condition **!c.2)** gives ap(f2) ≠ ap(fk), which excludes condition **a)** of proposition 7 for the fault [f2, fk]. From **(1)** we obtain vp(f1) ≠ vp(f2) and vp(f1) ≠ vp(fk). Thus, ap(f2) = vp(f1) in **(3.2.1)** gives ap(f2) ≠ vp(f2) and ap(f2) ≠ vp(fk). For the fault [f2, fk], these relations exclude the conditions **b.1)** and **b.2)**.

In case **(3.2.2)**, ap(f1) ≠ ap(fk) invalidates condition **a)** of proposition 7 for the fault [f1, fk]. From (1) we obtain vp(f1) ≠ vp(f2) and vp(f2) ≠ vp(fk). Thus, ap(f1) = vp(f2) in **(3.2.2)** gives ap(f1) ≠ vp(f1) and ap(f1) ≠ vp(fk). For the fault [f1, fk], these relations exclude the condition **b.1)** and **b.2)**.

Therefore, in all possible situations concerning the case **!c.1 and !c.2,** there is at least a double fault **(**[f1, f2], [f2, fk], or [f1, fk]), which does not satisfy neither condition a) nor condition b) of proposition 7. Thus, proposition 8 is proven for the case **!c.1 and !c.2**.

[f1, f2, … fk] satisfies c.2*:* Let us consider the double faults [f1, f2], [f1, fk], and [f2, fk]. Let a1, a2, and ak be respectively the *aggressor-sensitizing states* of f1, f2, and fk. Since a1, a2, and a3 take binary values (0 or 1), two of them will necessarily have equal values. Let, for instance a1 = ak. Furthermore, since condition **c.2** is satisfied, the aggressors of f1 and fk have the same bit positions. Then, from the proof of proposition 7, f1 and fk will be detected by a single read. This implies that the *single-read double-fault detection* property is satisfied when condition **c.2** is true.

[f1, f2, … fk] satisfies c.1. Let us consider the double faults [f1, f2], [f1, fk], and [f2, fk]. Let a1, a2, ak be respectively the *aggressor-sensitizing states* of f1, f2, fk and v1, v2, vk be their *victim-sensitizing states*. Since a1, a2, ak, v1, v2, and vk take binary values (0 or 1), then, all possible cases concerning the variables a1, v1, a2 and v2 are:

- a1= v1 and a2=v2. As in case **c.1** the aggressor cell of each fault has the same bit position as the victim cell of the fault, then, from the proof of proposition 7 f1 and f2 will be detected by a single read.
- a1≠ v1 and a2≠v2. In this case again, from the proof of proposition 7, f1 and f2 will be detected by a single read.
- a1= v1 and a2≠v2. Then we can have either ak = vk or ak≠ vk. From the proof of proposition 7, in the first case f1 and fk will be detected by a single read, and in the second case f2 and fk will be detected by a single read.
- a1≠ v1 and a2=v2. This case is treated similarly to the previous one.

Therefore in all possible cases of condition **c.1**, f1 and f2, or f1 and fk, or f2 and fk are detected by a single read. Thus, from lemma 2 the *single-read double-fault detection* property is satisfied when condition **c.1** is true. **QED**

The outcome of lemma 16 is important as it shows a 100% coverage for faults of multiplicity higher than 2. Its impact is even wider, as it enables 100% coverage for faults of any multiplicity by means of a simple and low-cost approach proposed next.

2.4 TREATMENT OF SRDF DECEPTIVE FAULTS

The proposed march tests *achieve the single-read double-fault detection property for all faults of multiplicity higher than two, as well as for the vast majority of faults of multiplicity 2*. We have also shown (Lemma 15), that these faults cannot be covered by any march tests (***march-SRDF-deceptive* faults**). The non-covered faults consist in double faults [f1, f2] having the following characteristics:

- f1 and f2 are of the type CFtr or CFwd (giving 4 double fault combinations), or f1 and f2 are of the type CFir (giving 1 double fault combination). Considering that there are 15 single-cell and two-cells FFMs, which give 225 possible double faults combinations (15 options for f1 x 15 options for f2), the non-covered double faults represent 2.2% of all possible double faults combinations (5/225).
- The bit positions of the two aggressor cells coincide, or the bit positions of the two aggressor cells coincide with the bit positions of the two victim cells. For a memory using m-bits words, there are m^4 distributions of the bit positions of the aggressor and victim cells. The distributions in which the bit positions of the aggressor cells coincide are m^3. The distributions in which the bit positions of the aggressor cells coincide with the bit positions of the victim cells are $2m^2$. Thus, the proportion of bit-positions distributions leading to non-covered faults is $(m^3 + 2\ m^2)/\ m^4 = (m+2)/m^2$

- The sensitizing quadruplets leading to non-covered faults correspond to the half of the total number of sensitizing quadruplets.

From this analysis, the percentage of double faults that are not covered by the march test algorithms proposed in this chapter (and by any march test - ***march-SRDF-deceptive*** **faults**) is equal to $(5/225)x(m+2)/m^2x0.5x100\% = 10(m+2)/9(m^2)\%$. For memory word size of 22-bits (16 data bits and 6 SECDED check bits), and 39-bits (32 data bits and 7 SECDED check bits), this percentage is respectively equal to 0.055%, and 0.03%. Thus, they represent a very small fraction of a particular case of faults (double-faults affecting the same memory word), and may be ignored in many applications. However, if the application requires covering them, we can employ one of the following approaches.

A first approach consists in developing test algorithms dedicated to this small fraction of faults. To cover these faults we need to supply different values to the same bit positions of different words (in order to address the issue of bit positions coincidences), which is not possible with march tests. To cope with this issue we can use non-march algorithms that have the same simple structure as march algorithms (to preserve simple BIST implementation), but instead of using inside each march element the same vector V_i for all memory words, we can use shifted versions of V_i. The design of such algorithms is the subject of future developments.

A second approach exploits the fact that, as shown in proposition 8, *all faults affecting the same memory word and having multiplicity higher than 2, are covered by the proposed algorithms*. Thus, as all non-covered faults produce errors that are detectable by ECC, we can employ the following strategy: These faults are not detected during the test and repair phase and are left unrepaired. Then, each time a double error is detected by the ECC during system operation, the affected memory word is replaced by a good word of the repair CAM. This strategy is challenged by two potential problems:

- Repairing the memory words producing double errors during system operation will imply repairing not only those comprising double faults but also those containing a single faulty cell and are affected by an SEU. This will result in wasting spare resources. However, from the analysis in the section 1.1 of chapter 1, in a very large memory of 100 Gbit and for a very high defect density of 10^{-3}, this will happen for less than 2 times per three years. Thus, in a ten years life of a 100 Gbit SRAM, the wasted spare words will be less than 7. Adding 7 extra words in the CAM used for repair, represents an insignificant percentage of area and power cost with respect to a 100 Gbit SRAM.
- If a soft error affects such a memory word W before it is detected by the ECC, (and hence repaired as proposed above), it may induce a triple error, which is not detected by the ECC. However, the probability of this kind of event is extremely low. Indeed, such an event will occur if an SEU affects at runtime the word W during an opportunity-time-window [t1 t2] such that:
  i. At t2 word W is read.
  ii. At t1 a write operation is performed over W and no other write is performed over W between t1 and t2.
  iii. The double fault affecting W is sensitized during the time-window [t1, t2].
  iv. [t1, t2] is the very first time-window satisfying conditions i., ii., and iii., and happening after the occurrence of the double fault (which can occur during manufacturing or in the field). This is because, if no SEU occurs on W during this very first opportunity-time-window, then, as the double fault is sensitized during [t1, t2], the read operation at t2 will detect a double error and the faulty word W will be repaired.

The duration of [t1, t2] determines the occurrence probability of the triple error. This duration depends on the application program, and can be evaluated through application simulations (e.g. of benchmark software applications), by taking into account the sensitizing conditions of the *march-SRDF-deceptive* faults. However this kind of evaluation is not in the scope of this work. Instead we will consider a very pessimistic

value for the mean duration of [t1, t2], such as one week! Considering this pessimistic value for [t1, t2], and a very large memory of 100 Gbit capacity, using words of 32 data bits and 7 SECDED check bits, and affected by a high defect density 10^{-3}, we find that the number of triple undetectable errors that will occur during the full life of a product is equal to $(0.999^{37}x10^{-3}x10^{-3}x19x39)x(0,0003)x(37/39)x(10^{11}/10^{6})$ $x(8+2x6)x(10^{9}x7\ x24)$. This gives $7.1X10^{-8}$ events for the full circuit life, which is extremely low!

2.5 EVALUATIONS

CAM size, area and power penalties

To evaluate the power and area improvements of the proposed approach, we determine the size of the CAM required for conventional repair and for ECC-based repair, in order to achieve 97% yield. Then, we use CACTI [34][56][57] to evaluate the area and power cost of the SRAMs under repair and the repair CAMs.

We can compute the yield for both conventional repair and ECC-based repair by using the following analytical expression (see chapter 5 for a more detailed explanation):

$$Y = \sum_{t=0}^{N_{wc}} \left(\frac{N_w! P_{wg}^{(N_w - t)}}{(N_w - t)!\, t!} (1 - P_{wg})^t \sum_{r=0}^{N_{wc}-t} \frac{N_{wc}! P_{wcg}^{N_{wc} - r}}{(N_{wc} - r)!\, r!} (1 - P_{wcg})^r \right) \quad (1).$$

As determined in chapter 5, the number of operations required to compute the yield by means of (1) is: $Nw(Nwc + 1) + (Nwc^2 - 1)(5Nwc + 12)/6 + 1$ multiplications; $(Nwc + 1)(Nwc + 4)/2$ divisions; and $N_{WC}(N_{WC} + 3)/2$ additions, where Nw is the number of words of the memory under repair, and Nwc is the number of words of the repair CAM. The computation complexity is much higher with respect to the yield computation for low defect densities, where faults affecting the CAM have insignificant impact to the yield and are ignored, resulting in much simpler yield computation expression. Also, as we deal with future very advanced technologies allowing producing very complex chips, we should be able to deal with very large memories. Moreover, as we deal with high defect densities we must be able to deal with large repair CAMs. In this context, the above numbers of operations are too large. Furthermore, these operations have to manipulate very large numbers as well as very small numbers, requiring high precision arithmetic. Thus, computing the yield by means of expression (1) becomes computationally intractable. To accelerate these computations we discovered certain recursive relations described in chapter 5, which reduce the number of operations at linear complexity, resulting in only: $N_W + 8N_{WC} - 1$ multiplications, $2N_{WC}$ divisions, $2N_{WC}$ additions, and N_{WC} subtractions. This is dramatically shorter with respect to the number of operations required for computing expression (1) in direct manner.

The new recursive yield computation approach was implemented in C++, and we used this tool to determine the size of the repair CAM, for several defect densities and for two SoCs comprising each a total of 9,75 Gbit SRAM capacity (i.e. a total of 250M words x 39 bits per word, corresponding to 32 data bits and 7 Hamming code bits). In the one SoC this memory capacity is distributed over 300 embedded memories, and in the other SoC this memory capacity is distributed over 3000 embedded memories. We also used CACTI to evaluate the area and power penalties for non-ECC repair and the ECC-based repair approaches. The results are presented in table 5, where the target yield for the total memory capacity of each SoC is 90%.

In this table, column 1 gives the considered defect density (expressed as the probability of a memory cell to be faulty), and column 2 gives the number of the embedded memories over which the total SRAM capacity of 9,75 Gbit is distributed.

Columns 3, 4, and 5 provide the results for conventional (i.e. Non-ECC) repair: column 3 presents the number of CAM words required to reach the target yield (i.e. 90%), columns 4 and 5 give the area and the power penalties.

Columns 6 to 8 provide the results for ECC-based repair: column 6 presents the number of CAM words

required to reach the target yield (i.e. 90%), columns 7 and 8 give the area and the power penalties.
From these results we observe that ECC-based repair achieves drastic reduction of both area and power penalties.

Table 5. Area and power cost comparison

P_f	Embed. Mem	Conventional Repair			ECC-based Repair		
		N_{CW}	%A	%P	N_{CW}	%A	%P
10^{-4}	300	3466	1.32	185.3	16	0.008	1.267
	3000	402	1.27	67.90	6	0.028	1.297
3x10 -4	300	10285	3.93	532.9	83	0.036	5.337
	3000	1121	3.46	177.9	17	0.078	3.676
10^{-3}	300	35325	12.75	1629	720	0.249	39.56
	3000	3693	13.49	581.5	98	0.344	17.56

Test Duration

The length of the march tests proposed in section 2.2 is equal to 31x(size of the 2-covering set) + 7x(size of the 3-covering set) + 4x(size of the 4-covering set). For 39-bits memory words (32 data bits and 7 ECC check bits), the size of a 2-covering sets is 14, and the size of the 3-covering and 4-covering sets proposed in [32][33][58] is respectively 25 and 66. Thus, for memories employing 32-bit words plus 7 check bits, the test length is equal to 31x14 + 7x25 + 4x66 = 434 + 175 + 264 = 873 Nw, where Nw is the number of memory words. This is 46 times longer than the length of the conventional (i.e. non-SRDF) test algorithm proposed in [31], which is an optimal algorithm covering the FFMs of tables 1 and 2. However, the comparison of the discussed approaches should consider the test cost, which is determined by the test duration rather than the length of the test algorithm.
As the power dissipation during the test and repair sessions is larger for the conventional repair approach, its test duration is increased proportionally, because the permitted power dissipation will allow testing a proportionally lower number of memory blocks. The conventional test algorithm is used in the case of the conventional repair, while the SRDF test algorithm is used in the case of the ECC-based repair. Then, using the power dissipation results of table 5 we find the test time increase for the SRDF algorithm. The results are presented in table 6.
In this table, column 1 gives the defect density, column 2 gives the number of embedded memories over which the total SRAM capacity of 9,75 Gbit is distributed; column 3 gives the total power of the memory system for conventional repair (i.e. the power of the SRAM under repair plus the power of the repair CAM used for conventional repair) divided by the power of the SRAM; column 4 gives the total power of the memory system for ECC-based repair (i.e. power of the SRAM under repair plus power of the repair CAM used for ECC-based repair) divided by the power of the SRAM; and column 5 gives the increase of test time for the ECC-based repair, determined as 46x(total power of ECC-based repair)/(total power of conventional repair), that is 46x(column 4)/(column 3), where 46 is the number of times the SRDF algorithm is larger than the conventional test algorithm.

Table 6. Test-time increase

P_f	#Embedded Memories	Conventional Repair	ECC-based Repair	
		Total Power	Total Power	Test-time increase
10^{-4}	300	2.85	1.013	16.35

	3000	1.68	1.013	27.7
$3x10^{-4}$	300	6.33	1.053	7.65
	3000	2.78	1.037	17.15
10^{-3}	300	17.29	1.395	3.71
	3000	6.81	1.175	7.94

We observe that even in the worst case scenario for the proposed approach, i.e. for Pf = 10^{-4} and for 3000 embedded memories - where the test-time increase in table 6 takes its largest value (27.7) and the power penalty of the conventional repair takes its smallest value (67.9%) - the proposed approach is clearly more attractive as this power penalty is totally inacceptable. Furthermore the situation turns in the decisive advantage of the proposed approach as the defect density increases. Thus, for Pf = 10^{-3} and 300 embedded memories the test time is increased by a factor of 3.71, which is clearly preferable than the huge 1629% power increase induced by the conventional repair approach. Also, the 12.75% area penalty induced by the conventional approach is totally undesirable. Indeed, as memories occupy the largest part of modern SoCs (more than 90% of the SoC area in most cases), the 12.75% area penalty induced by the conventional repair approach represents more than 11.5% of the total SoC area.

2.6 CONCLUSION

ECC-based repair is the only known solution that can cope with high defect densities at reasonable cost. However, in chapter 1 we have found that, due to fault-diagnosis issues, ECC-based repair may lose its advantages (the fault-diagnosis process requires a CAM of the same size as the CAM used in conventional repair). In chapter 1 we have also proposed 3 approaches for solving this problem. One of them introduces a new family of memory test algorithms (referred to as single-read double-fault detection – SRDF – algorithms), which detect in a single read at least two faulty cells in any memory word containing more than one faulty cell. Thanks to this property, SRDF test algorithms completely eliminate the diagnosis hardware, leading in dramatic cost reduction. Thus, the aim of this chapter is the development of SRDF test algorithms for a comprehensive fault model including all single-cell and all two-cell static unlinked faults. However, developing test algorithms satisfying the SRDF property is far more complex in comparison with the development of conventional test algorithms, because the number of fault cases and their complexity increase dramatically.

In this chapter we addressed successfully this highly complex theoretical challenge. In particular, we developed a theoretical framework for treating this challenge, and based on this framework, we derived march tests for all static unlinked functional fault models (FFMs) involving one memory cell (single-cell FFMs) and two memory cells (two-cell FFMs). For this comprehensive fault model, the proposed march tests *achieve the single-read double-fault detection property for all faults of multiplicity higher than two, as well as for the vast majority of faults of multiplicity 2*. We have also shown that, the few double faults that are not covered by our test algorithms cannot be covered by any march test algorithms. However, based on the results of our theoretical analysis of the fault coverage of the SRDF test algorithms, we also proposed a simple and low-cost protection for the non-covered faults, allowing 100% protection for any fault multiplicity.

To evaluate the proposed approach at reasonable computation time, we used a yield computation tool developed in chapter 5, which achieves dramatic acceleration of the yield computation and enables us coping computing the yield for large memories and high defect densities. Thanks to this tool, we determined the CAM size for conventional repair and for ECC-based repair using SRDF test algorithms, by considering various fault densities and SRAM systems. Then, we used CACTI to estimate the area and power penalties

of these approaches.

These evaluations show that the proposed approach based on SRDF test algorithms achieves dramatic reduction of area and power penalties with respect to conventional repair. This reduction fully justifies an extra cost in test duration. These results are completed in chapters 3 and 4 with:

- An approach using separate diagnosis CAM and runtime CAM, to reduce runtime-power penalty with respect to the conventional repair (i.e. the same as ECC-based repair using SRDF test algorithms), but as this scheme induces very high area penalty (similar as conventional repair), we also developed and iterative diagnosis scheme that reduces the size of the diagnosis CAM at the expense of extra test time.
- A low-power word-repair architecture, further reducing runtime power.

These developments provide a comprehensive framework enabling very low power penalty and efficient trade-offs in terms of test time and hardware cost.

CHAPTER 3

ITERATIVE DIAGNOSIS APPROACH FOR ECC-BASED MEMORY REPAIR

As highlighted in chapter 1, ECC-based memory repair reduces dramatically the repair area and power costs in technologies affected by high defect densities, but these gains can be lost as the diagnosis hardware can be as complex as the hardware of conventional repair. To resolve this problem, in chapter 2 we proposed and developed a new family of test algorithms that exhibit the single-read double-fault detection (SRDF) property. These algorithms eliminate completely the diagnosis hardware, leading in dramatic reduction of area and power cost. However, this is achieved at the expense of significant extra test time. In order to dispose a second alternative, in this chapter we propose and explore a scheme using two separate CAMs: a small CAM used for ECC-based repair (repair-CAM), and a large CAM used for diagnosis (diagnosis-CAM). This scheme uses conventional test algorithms. Thus, the test length is much smaller than in the case of SRDF test algorithms. Also, as only the small repair-CAM is used at runtime, runtime power is reduced dramatically with respect to conventional repair. However, the area cost is very high as the diagnosis-CAM is as large as the CAM of conventional repair. Then, to reduce this cost, we also propose an approach, which uses smaller diagnosis-CAM and compensates the reduced CAM space by executing the test algorithm multiple times and diagnosing at each iteration a subset of the faulty memory words. This iterative approach allows trade-offs between the diagnosis-CAM size and the test length (number of iterations). Thus, together with the SRDF test algorithms, which achieve the lowest area and power cost but maximum test length, it provides a comprehensive framework for area, power, and test length trade-offs. The rest of this chapter deals with the *separate-CAMs* scheme and the iterative diagnosis approach.

3.1 SEPARATE-CAMS SCHEME FOR RUNTIME POWER REDUCTION

As highlighted in chapter 1, if we do not employ SRDF test algorithms we need to store in the CAM during diagnosis all faulty memory words, in order to determine those comprising more than one faulty cell. On the other hand, for repair purposes we need to store in the CAM only the words comprising more than one faulty cell. Thus, to reduce runtime power we use a large CAM for diagnosis purposes (diagnosis-CAM). Then, at the end of the test and diagnosis phase we transfer the memory words comprising multiple faulty cells in a separate small CAM (runtime-CAM). At runtime, only this CAM is used and powered. As it is drastically smaller than the diagnosis-CAM (which has the same size as the CAM used in conventional repair), it consumes drastically lower power than conventional repair. Note also that, though the diagnosis-

CAM of this scheme has the same size as the CAM used in conventional repair, it may also allow significant area reduction with respect to conventional self-repair. Indeed, after fabrication, the diagnosis information has to be stored in non-volatile embedded memory, to guarantee that faulty memory locations detected and diagnosed by thorough fabrication tests are permanently stored in the chip after power shutdown. Conventional self-repair will save in non-volatile memory all faulty memory words, while, in ECC-based repair, only words comprising multiple faulty cells will be stored in non-volatile memory. Thus, *drastically smaller non-volatile memory is required*.

Let us now describe the operation of the *separate-CAMs* scheme.

Each location of the diagnosis-CAM and the runtime-CAM is composed of a tag field in which we store the addresses of faulty words of the memory under repair, and a data field in which we store the positions of the faulty cells of these words. Each CAM location also possesses a flag cell (flag1) used to indicate *bad* CAM locations. *In both, the diagnosis-CAM and the runtime-CAM the flag1 cells are initialized to 0, and during the test sessions of these CAMs*, flag1 is set to 1 in each *bad* CAM location (i.e. a CAM location that cannot be used for repair because it contains faults in the tag field or in the flag cells, or more than one faulty data cells). If a flag1 cell is faulty, it may indicate as *good* for performing repair a bad CAM word. To avoid this problem, a second flag cell (flag1') can be added in each CAM location [19]. Another flag cell (flag2) is used to indicate that the CAM location is not free (i.e. it is occupied by information concerning a faulty memory word).

Let us now discuss the operation of the diagnosis-CAM during *the test and diagnosis of the memory under repair*. When the current read operation of the test algorithm detects a faulty memory word, the contents of the diagnosis-CAM have to be updated. Thus, the address of the faulty memory word is compared with all the tag fields of this CAM. This comparison is performed by a comparator integrated in each tag field. The tag field also includes circuitry that deactivates the match signal if flag1 = 1 or flag2 = 1. Thus, *a hit occurs* only if: the tag comparison matches (meaning that the faulty address is already stored in the tag field of a CAM location); flag1 = 0 (meaning that the CAM location is *good*); and flag2 = 0 (meaning that the CAM location is occupied by diagnosis information of a faulty memory word). Then, two CAM-updating mechanisms are used (Hit-updating and Miss-updating).

- *Hit-updating:* In case of hit, the activated match line selects the data field of the hit CAM location, in which we update the positions of the faulty cells of the memory word. This update is done by: reading the content of the selected data field; bit-wise ORing it with a vector containing the faulty-cell positions of the faulty memory word; and writing the result back to the selected data field. Note that the vector containing the faulty-cell positions of the faulty memory word is obtained from the current outputs of the XOR gates of the BIST comparator.
- *Miss-updating*. In case of miss, a free CAM location has to be selected for storing the faulty address and the positions of the faulty cells. This selection mechanism uses a counter (to be referred as *FLC* – free-locations counter), whose content identify a free CAM location. The content of *FLC* is decoded to activate the word-line selecting the tag field and the data field of this location[6]. Then, the address of the faulty word is written in the selected tag field, the vector containing the faulty-cell positions is written in the data field, and the value 0 is written in the flag2 cell to indicate that the CAM location is occupied (it contains diagnosis information of a faulty memory word).

Before starting the test and diagnosis phase of the memory under repair, the tag fields and the data fields of the diagnosis CAM are set to 0, and the flag2 cells are set to 1. The *FLC* counter is also reset. If the first CAM location selected by *FLC* has flag1 = 1 *FLC* increments, and this is repeated each time the next location has flag = 1.

[6] Alternatively a shift-register containing 1 in one position and 0 in all other positions can be used instead of the counter *FLC* and the decoder. Each output of this register activates one word-line of the CAM.

Test and diagnosis process:
During the test and diagnosis phase of the memory under repair, each time a fault is detected in this memory, the diagnosis-CAM is updated by means of the *Hit-updating* or the *Miss-updating* mechanism as described above. In addition, when *Miss-updating* is activated, *FLC* is incremented. If the new location pointed by *FLC* has flag1 = 1, then *FLC* increments again. Thus, *FLC* always points a *good* CAM location (which is also unoccupied because diagnosis information is stored only in locations already visited by FLC).

At the end of the test and diagnosis phase, *FLC* is used to visit and read each location of the diagnosis-CAM. For each diagnosis-CAM location having flag1= 0, flag2 = 0, and containing multiple 1's in the data field, the tag field is transferred to a *good* location of the runtime-CAM[7] (and to the non-volatile memory if any), and the flag2 cell of this location of the runtime CAM is set to 1. To select sequentially the locations of the runtime-CAM during this transfer, we employ a mechanism using a counter similar to *FLC*.

Runtime operation:
Similarly to the diagnosis-CAM, each location of the runtime-CAM includes a comparator and extra circuitry that deactivates the match signal if flag1 = 1 or flag2 = 1. Thus, a hit occurs in the runtime-CAM only if the tag comparison matches, flag1 = 0 (meaning that the CAM location is *good*), and flag2 = 0 (meaning that the CAM location is occupied by a faulty memory word). At runtime, reads and writes are performed over the memory, but, at the same time the runtime-CAM compares in parallel the address of the current memory operation with its tag fields. In case of hit: if the memory operation is a write, the data are also written in the data field of the hit CAM location; if the memory operation is a read, the data field of the hit CAM location is read and the read data are supplied to the data bus of the system. To do this, the hit signal controls a multiplexer. Then, each time this signal is active it disconnects the memory from the data bus and connects instead the runtime-CAM. As the runtime-CAM is small even for high defect densities, and it is accessed in parallel with the memory, it will not induce performance penalty. Thus, the only performance penalty is due to the MUX added in the data bus. On the contrary, conventional repair in high defect densities will require very large CAM, which will induce non-negligible performance penalty.

3.2 ITERATIVE DIAGNOSIS FOR DIAGNOSIS-CAM REDUCTION

To reduce the cost of the diagnosis hardware we will use a smaller diagnosis-CAM, which could not store the addresses of all faulty memory words. To compensate the missing diagnosis-CAM capacity, we will execute the test algorithm several times. After each iteration of the test algorithm we will free a part of the CAM to create space for treating new faults. This process could reduce fault coverage. Taking into account this issue, the proposed approach works as follows.

At each iteration ⌶ of the test algorithm:

i. Associative search followed by *Hit-updating* or *Miss-updating* is performed each time a read operation of the test algorithm detects a faulty memory word, as described in section 3.1.
ii. If there are not available *good* CAM locations[8] and a new miss occurs due to a read detecting a faulty memory word , not yet stored in the CAM, then, the updating of this miss cannot be realized (*aborted Miss-updating*). In this case, we store in a register (to be referred as CR register) the read cycle of the test algorithm in which the *aborted Miss-updating* occurs (say test *cycle ci*). This cycle can be

[7] In the field, after the first test and diagnosis phase the tag fields and the data and fields of the runtime-CAM are set to 0, and its flag2 cells are set to 1. Subsequently this may not be done if we wish to maintain in the runtime CAM faulty words stored during the previous test and repair phases.

[8] This means that: a read operation has detected a faulty memory word not yet stored in the diagnosis-CAM; the Miss-updating was realized for this word by storing it in the available fault-free CAM location pointed by *FLC*; and *FLC* was incremented until the last diagnosis-CAM location but no location with flag1=0 was found due to insufficient CAM space.

identified by using a counter that is incremented at each read operation performed by the test algorithm, or by the states of the BIST-hardware counters (e.g. in the case of march test algorithms, the counter identifying the current march element, the address counter, and the counter identifying the current operation executed in the address pointed by the address counter).

iii. After the *aborted Miss-updating* we continue executing the current iteration of the test algorithm and performing *Hit-updatings*, but we stop performing *Miss-updatings*. Thus, starting at c*ycle ci* and until the end of the test algorithm, we only update the faulty-cell positions of faulty memory words already stored in the diagnosis-CAM, but we do not store (due to lack of CAM space) the addresses and the faulty-cell positions of any other faulty words (i.e. faulty words that are detected for first time at *cycle ci* or after it).

iv. At the end of the current iteration of the test-algorithm, each *good* diagnosis-CAM location whose data field contains exactly one "1" is cleared to create available CAM locations for the next test-algorithm iteration, while the *good* CAM locations containing multiple "1"s in their data field are rearranged to occupy the lower addresses of the CAM. To realize these arrangements we use two counters: the counter *FLC* used above and a second counter *FLC2*. First we reset both *FLC* and *FLC2*. Then, *FLC* and *FLC2* are incremented until finding the first CAM location storing a word having flag1=0 and containing exactly one "1" in its data field. At this location *FLC* stops incrementing, while *FLC2* increments until finding a CAM location having flag1=0 and containing more than one "1" in its data field. The contents of the latter location are transferred to the former location. Then, *FLC* increments until the next location storing a word having flag1=0 and containing exactly one "1" in its data field, while *FLC2* increments until the next location having flag1=0 and containing more than one "1" in its data field. Again, the contents of the latter location are transferred to the former location, and so on until *FLC2* reaches the last location of the diagnosis-CAM. *FLC* increments to select the subsequent CAM locations and each selected location is cleared (i.e. its tag and data fields are set to 0, and the flag2 cells are set to 1). This process stops when the last CAM location is cleared. If at test *cycle ci* the CAM stores r words that contain multiple faulty cells, at the end of this process these words will be stored in the first q CAM locations comprising r *good* locations, and the remaining locations (i.e. those identified by states of *FLC* larger than q-1), will be either faulty (flag1 = 1) or free to use in the next test iteration (flag1 = 0 and flag2 = 1).

In the next iteration $]+1$ we re-execute the whole test algorithm (i.e. from its beginning until its end), but we do not update the CAM until reaching the test *cycle ci* (saved in the *CR* register as stated earlier). Starting from the test *cycle ci* we update the CAM in the similar way as before, except that in this iteration the address counter *FLC* starts incrementing from state q. The iterations of the test algorithm are repeated until one of the following stop conditions occurs:

S.1 In some iteration the end of the test algorithm is reached and no *aborted Miss-updating* has occurred (the iterative diagnosis process ends successfully).

S.2 In some iteration an *aborted Miss-updating* has occurred and no CAM location having flag1=0 and containing exactly one "1" in its data field, has been found in step iv by *FLC2* (the iterative diagnosis process fails).

The following propositions demonstrate the pertinence of our iterative diagnosis approach.

Proposition 1: The proposed iterative diagnosis approach finishes within a finite number of iterations.

Proof: The following statement is always true: the iterative diagnosis approach fails within a finite number of iterations, or the iterative diagnosis approach never fails. Let us consider each of the two possible cases of this true statement.

1st case of the true statement: the iterative diagnosis approach fails within a finite number of iterations. This means that, within a finite number of iterations, an *aborted Miss-updating* occurs and no CAM location

having flag1=0 and containing one "1" in its data field has been found in step iv. In this case, the stop condition S2 is reached in a finite number of iterations. Thus, the iterative diagnosis algorithm stops in a finite number of iterations.

2nd case of the true statement: the iterative diagnosis approach never fails. This case implies that one of the cases *a* or *b* is true:

- ***a.*** *Aborted Miss-updating* never occurs, or
- ***b.*** Each time an *aborted Miss-updating* occurs, there is at least a CAM location having flag1=0 and containing exactly one "1" in its data field.

If case *a* is true then: as the memory test algorithm has finite length, we reach its end in finite time. Furthermore, as none of the Miss-updatings is aborted, the stop condition S1 is reached at the end of the test algorithm.

If case *b* is true then, let us consider an *aborted Miss-updating* occurring at a test *cycle ci* of iteration I. In this situation, case *b* implies that at the test *cycle ci* there are some CAM locations having flag1=0 and containing exactly one "1" in their data field. Then, one of the following two cases concerning these locations will occur:

- **b.1** When the test algorithm reaches its end during iteration I, the data fields of all these locations have been updated and contain more than one "1" in their data fields. This also implies that no CAM location is cleared at the end of iteration I, so that there are not available CAM locations.
- **b.2** When the test algorithm reaches its end during iteration I, one or more of these locations contain(s) exactly one '1" and is (are) cleared after the end of the test algorithm of iteration I.

Furthermore, the occurrence of an *aborted Miss-updating* at a test *cycle ci* of an iteration I means that a faulty memory word (Wn) not yet stored in the CAM is detected at the test *cycle ci* of iteration I, and there is not available CAM location for storing it. Thus, in case b.1, when we reach the test *cycle ci* at the next iteration ($I+1$) all the following conditions are realized:

i. The faulty word Wn is not stored in the CAM.
ii. There are not unoccupied locations in the CAM.
iii. There are not CAM locations having flag1=0 and containing exactly one "1" in their data fields.

Conditions i and ii imply that the detection of the faulty word Wn in the test *cycle ci* of iteration $I+1$ will lead to an *aborted Miss-updating*. But as at the same time condition iii implies that there are no CAM locations having flag1=0 and containing exactly one "1" in their data fields, case b is contradicted. Thus case b always implies b.2, which means that case *b* always implies that one or more CAM locations are cleared at the end of iteration I. By the construction of our iterative diagnosis process, in the next iteration ($I+1$) we start updating the CAM at the test *cycle ci*. As one or more CAM locations were cleared at the end of iteration I, and at iteration $I+1$ we start updating the CAM at the test *cycle ci*, then, when the faulty word Wn is detected again at the test *cycle ci* of iteration $I+1$, there is at least one available CAM location, and the faulty word Wn is stored in such a location. Thus, in iteration $I+1$ *aborted Miss-updating* can occur only after the test *cycle ci*. Therefore, case *b* implies that: if an *aborted Miss-updating* occurs in any test *cycle ci* of any iteration I, then in the next iteration $I+1$, *aborted Miss-updating* can occur only after the test *cycle ci*. Therefore, in any iteration J subsequent to I: either no *aborted Miss-updating* occurs, and in this case at the end of iteration J the stop condition S1 is realized; or in iteration J an *aborted Miss-updating* occurs in a test *cycle cj*, and in iteration $J+1$ *aborted Miss-updating* can occur only after the test *cycle cj*. Thus, as long as the iterative diagnosis process does reach the stop condition S1, the test cycle in which *aborted Miss-updating* is occurring, increases. As the number of the test cycles is finite (finite test algorithm length), then, within a finite number of steps we will reach an iteration in which no *aborted Miss-updating* occurs,

implying the realization of the stop condition S1.
Thus, in all possible cases, the stop condition S1 or the stop condition S2 is reached in a finite number of iterations. **QED**

An important question concerns the condition, which ensures that our iterative diagnosis finishes successfully. This question is addressed in the following proposition.

Proposition 2: If the number *r* of *good* CAM locations is larger than the number *t* of memory words containing multiple faults (i.e. $r > t$), the diagnosis process ends successfully and in finite time.

Proof: Let us suppose that condition S2 occurs in some iteration]. This means that two conditions are realized at test *cycle ci* of the iteration]:

i. an *aborted Miss-updating* has occurred, and
ii. there is not CAM location having flag1=0 and containing exactly one "1" in its data field.

Since an *aborted Miss-updating* has occurred (point i), then all the *r good* CAM locations are occupied by faulty memory words. As an occupied *good* CAM location contains in its data field either exactly one "1" or more than one "1", we have $r = na + nb$ (where *na* is the number of *good* CAM locations containing exactly one "1" in their data field, and *nb* is the number of *good* CAM locations containing more than one "1" in their data field). In addition, each *good* CAM location containing more than one "1' in its data field corresponds to a faulty memory word containing multiple faults. Thus, *nb* can never exceed *t* (i.e. $nb \leq t$). From $r = na + nb$ and $nb \leq t$ we have $na \geq r - t$. From the statement of the proposition we also have $r > t$. Thus we obtain $na > 0$. This means that there is at least one *good* CAM location containing exactly one "1" in its data field, and contradicts condition ii. Thus, conditions i, and ii cannot both be realized in an iteration. Thus, the stop condition S2 is never realized. As from proposition 1 in our iterative diagnosis process either the stop condition S1 or the stop condition S2 is realized within a finite number of iterations, the stop condition S1 will be realized within a finite number of iterations. **QED**

When the diagnosis-CAM is full we liberate *good* CAM locations in which the data field contains only one "1" (corresponding to one faulty cell), in order to create free space for storing subsequently memory words containing more than one faulty cell. However, a memory word in which one faulty cell was detected until a test cycle *ci*, may also contain faulty cells that are detected at a later test cycle. Thus, CAM-locations clearing may lead in misdiagnosing certain memory words containing more than one faulty cell. The following proposition shows that our iterative diagnosis process does not lead to misdiagnosis. In addition, its proof highlights the reasons leading to the proposed iterative diagnosis process.

Proposition 3: Let us suppose that the number of *good* locations of the diagnosis-CAM is larger than the number of memory words containing more than one faulty cell. Then, the quality of the diagnosis is not affected by the proposed iterative diagnosis process.

Proof: From proposition 1, the test algorithm stops successfully in finite number of iterations. Then, there are two issues that could affect diagnosis quality: reduction of fault coverage; loss of diagnosis-sensitive information.

Fault coverage reduction: In any iteration]+1, faults detected by the test algorithm are not used to update the CAM until the test *cycle ci* in which an *aborted Miss-updating* has occurred in the previous iteration]. Then, starting in iteration]+1 the execution of the test algorithm at test *cycle ci* would help reducing test time. However, fault sensitizations produced during the non-executed test cycles (i.e. the ones preceding *cycle ci*) would be lost. This will prevent the detection of faults that could be detected at test *cycle ci* or after it, but are sensitized by operations performed before this cycle. As at each iteration our approach executes the test algorithm from its beginning, we do not experience fault sensitization reduction[9].

Loss of diagnosis-sensitive information: For faulty words never evicted from the diagnosis-CAM loss of

[9] This part of the proof explains also why at each iteration we start the test algorithm from its beginning.

diagnosis-sensitive information cannot occur. However, this could happen for faulty words that are evicted from the CAM. Such loss may occur only if:

i. The test algorithm detects a faulty cell C1 and stores the related information in a CAM location.
ii. This CAM location is cleared later.
iii. Another faulty cell C2 affecting the same memory word is detected by the test algorithm after this clearing.

If event ii occurs after i, and event iii occurs after ii, then, the CAM will not contain simultaneously the information concerning both faulty cells, and the memory word could not be identified to contain multiple cells. However, our approach avoids this issue as each of the events i, ii, and iii can occur but: each time the CAM is full (say at *cycle ci* of the test algorithm), we continue executing the test algorithm until its end, and we also update the data field of the words detected as faulty until the test *cycle ci*. Thus, for any word detected as faulty before the test *cycle ci* we collect information concerning the positions of its faulty cells until the end of the test algorithm, guarantying that all faulty cells of this word will be detected before the CAM location containing this faulty word is cleared (i.e. event iii will occur between events i and ii). Thus, each word detected as faulty before the test *cycle ci* will be diagnosed correctly[10]. On the other hand, as the next iteration starts updating the CAM from *cycle ci*, then, any word detected as faulty for first time at the test *cycle ci* or after this cycle and until the next test cycle at which the CAM is again full (say test *cycle ck*), is correctly diagnosed for the same reasons as for the faults detected before cycle ci. The same reasoning holds for the words detected as faulty for first time at *cycle ck* or after this cycle and until the subsequent cycle at which the CAM is full, and so on until covering all faulty memory words. QED

From the proof of proposition 3, re-executing at each iteration the test algorithm from its beginning guaranties in all situations that there is no fault coverage reduction due to fault sensitization issues. However, corollary 1 allows reducing the length of certain iterations of the test algorithm.

Corollary 1: If an iteration I starts updating the diagnosis-CAM from an operation belonging to a test sequence S(i), then, iteration I can skip each sequence S(j) with j < i, which is not necessary for sensitizing the faults detected in S(i).

From corollary 1, with a dedicated analysis of the target test algorithm, it is possible to reduce the length of the iterative diagnosis approach. For instance, for the MSS1 algorithm presented in section 3.3, figure 2, if in an iteration I the *aborted Miss-updating* occurs in the sequence S(1), then, using proposition 3 we will have to start the next iteration $I+1$ from the beginning of the test algorithm. However, no-fault detected in sequence S(2) can be sensitized in sequence S(0). Thus, thanks to corollary 1, if in I the *aborted Miss-updating* occurs in the sequence S(2), we can start $I+1$ from the beginning of sequence S(1). Similarly, if in I the *aborted Miss-updating* occurs in the sequence S(3) (resp. S(4) or S(5)), we can start $I+1$ from the beginning of sequence S(2) (resp. S(3) or S(4)). Note however that, when we start iteration $I+1$ from the beginning of S(2) (resp. S(4)), we should pay attention before starting $I+1$ to initialize the memory by performing a W1 at each memory word (because as iteration I ends with the sequence S(5) the memory state is not compatible for starting executing S(2) nor S(4)).

Based on these observations, for the test algorithm MSS1 shown in figure 1, subsection 3.3.2.1, the reduction rl(i) of the number of operations for any iteration that starts updating the CAM at sequence S(i), $i \in \{1, 2, 3, 4, 5\}$ is: rl(1) = 0; rl(2) = N; rl(3) = 4N; rl(4) = 9N; rl(5) = 12N, where N is the number of memory words. We note that, the later an iteration starts updating the CAM, the higher the reduction of the

[10] This part of the proof explains also why after the occurrence of an *aborted Miss-updating* in any iteration of the diagnosis process, we continue the execution of test algorithm until its end, and we also continue updating until this end the faulty-cells positions of faulty memory words already stored in the CAM.

number of operations. Thus, for iterations starting updating the CAM at sequence S5 we have a 12N reduction, meaning that the length of these iterations will be 6N instead of 18N for the whole algorithm MSS1. This is advantageous because, as we progress towards the end of the iterative diagnosis process, the CAM locations are increasingly occupied leading in more frequent iterations that could have high impact on the diagnosis length. Fortunately, thanks to corollary 1, these iterations will have short duration.

3.3 AUTOMATION

3.3.1 Yield and CAM Size Computation

As discussed in chapter 2, we can compute the yield for both conventional repair and ECC-based repair by using the following analytical expression derived in chapter 5.

$$Y = \sum_{t=0}^{N_{wc}} \left(\frac{N_W! P_{Wg}^{(N_W - t)}}{(N_W - t)! t!} (1 - P_{Wg})^t \sum_{r=0}^{N_{wc} - t} \frac{N_{WC}! P_{WCg}^{N_{wc} - r}}{(N_{WC} - r)! r!} (1 - P_{WCg})^r \right) \quad (1)$$

where N_W is the number of memory words and N_{WC} is the number of the locations of the repair CAM, P_{Wg} is the probability that a memory word does not need to be repaired (*good* word), and P_{WCg} is the probability that a CAM location is good for repairing a faulty memory word (*good* CAM location).

Expression (1) is computed as the probability for a memory to be repaired successfully at runtime. For this repair to be successful, *the runtime-CAM must contain at least as many good locations as the number of memory words that need to be repaired*.

On the other hand, when we use separate diagnosis-CAM, we should also consider the probability for the diagnosis process to finish successfully. According to proposition 2, for the diagnosis process to finish successfully *the diagnosis-CAM must contain at least as many good locations as the number of memory words that need to be repaired* .We observe that the conditions for a memory to be repaired successfully at runtime and for the diagnosis process to finish successfully are identical. Thus, the expression (1) used to compute the memory yield as the probability for the memory to be repaired successfully, can also be used for computing the probability for the diagnosis process to finish successfully. However, what we are looking for is the joint probability for the memory to be repaired successfully and to be diagnosed successfully. This probability is given by the following expression:

$$Y = \sum_{t=0}^{N_{wc}} \left(\frac{N_W! P_{Wg}^{(N_W - t)}}{(N_W - t)! t!} (1 - P_{Wg})^t \sum_{r=0}^{N_{wc} - t} \frac{N_{WC}! P_{WCg}^{N_{wc} - r}}{(N_{WC} - r)! r!} (1 - P_{WCg})^r \sum_{r=0}^{N_{WDc} - t} \frac{N_{WDC}! P_{WDCg}^{N_{WDc} - r}}{(N_{WDC} - r)! r!} (1 - P_{WDCg})^r \right) \quad (2)$$

Where $\frac{N_W! P_{Wg}^{(N_W - t)}}{(N_W - t)! t!} (1 - P_{Wg})^t$ gives the probability that the memory contains Nw – t *good* words and t non-good words.

$\sum_{r=0}^{N_{wc} - t} \frac{N_{WC}! P_{WCg}^{N_{wc} - r}}{(N_{WC} - r)! r!} (1 - P_{WCg})^r$ **(3)** gives the probability that the runtime-CAM contains at least t *good* locations, and $\sum_{r=0}^{N_{WDc} - t} \frac{N_{WDC}! P_{WDCg}^{N_{WDc} - r}}{(N_{WDC} - r)! r!} (1 - P_{WDCg})^r$ **(4)** gives the probability that the diagnosis-CAM contains at least t *good* locations, and expression (2) gives the sum from t = 0 to t = N_{WC} of the product of these three probabilities, which provides the joint probability for the memory to be repaired successfully and to be diagnosed successfully.

Note that, as the number N_{WDC} of the locations of the diagnosis-CAM is larger than N_{WC}, the memory can be diagnosed successfully even if it contains more than t = N_{WC} non-good words. But in expression 2 the external sum is taken until the value t = N_{WC}. Hence, expression 2 does not consider the cases of successful

diagnosis in which the memory contains more than N_{WC} non-good words. However this is correct as: on the one hand expression (2) gives the joint probability for the memory to be repaired successfully and to be diagnosed successfully, and on the other hand if the memory contains more than N_{WC} non-good words, this joint probability is equal to 0 even though the memory can be diagnosed successfully (because the memory cannot be repaired when it contains more than N_{WC} non-good words).

We find that the number of operations required computing the yield by means of (2) is:

$N_{WDC}(N_{WC}+1)(N_{WC}+2)/2 + N_W(N_{WC}+1) + (N_{WC}+1)(7N_{WC}^2 + 2N_{WC} - 6)/6 -$ 3 multiplications; $(N_{WC}+1)(N_{WC}+3)$ divisions; and $N_{WC}(N_{WC}+2)$ additions. Thus the number of operations increases exponentially with the memory and the CAM sizes. In addition, we will have to deal with large values of N_W (as we have to deal with future very advanced technologies allowing producing very complex chips, which will include very large memories), as well as large values of N_{WDC} and N_{WC} (due to the high defect densities), leading to huge numbers of operations. Furthermore, these operations have to manipulate very large numbers such as Nw!/(Nw-t)!t!, as well as very small numbers such as P_{WCg}^{Nw}, requiring high precision arithmetic. Thus, computing the yield by means of expression (1) becomes computationally intractable.

To accelerate the computations, similarly as for expression (1), in chapter 5 we derive recursive relations for expression 5, which allow computing expression (2) in linear time.

Thus, thanks to the derived recursive relations, the computation of the yield is accelerated drastically, but its complexity is still non-linear. However, we observe that in expression (2) the terms

$$B_r = \frac{N_{wc}! P_{wcg}^{N_{wc}-r}}{(N_{wc}-r)!\,r!}(1-P_{wcg})^r \text{ and } C_t = \frac{N_w! P_{wg}^{(N_w-t)}}{(N_w-t)!\,t!}(1-P_{wg})^t$$

are used intensively. Thus, we compute them just once and we store them in two look-up tables for reuse during the computations. Then, we find that the yield computation can be done in linear complexity with respect to the memory size: $N_W + 9N_{WC} + 4N_{WDC}$ - 1 multiplications, $2N_{WC} + N_{WDC}$ divisions, $2N_{WC} + N_{WDC}$ additions, and $N_{WC} + N_{WDC}$ subtractions. This is dramatically shorter with respect to the number of operations required for computing expression (2) in direct manner.

Furthermore, using the recursive relations derived in chapter 5, allows also fast computation of expression (1), by means of $N_W + 8N_{WC}$ - 1 multiplications, $2N_{WC}$ divisions, $2N_{WC}$ additions, and N_{WC} subtractions.

Based on these developments, we implemented a tool enabling fast computation of the size of the runtime-CAM and the diagnosis-CAM required for achieving a target yield. In particular, we use the fast computation of expression (1) to compute the size of the runtime-CAM for conventional repair and ECC-based repair using SRDF algorithms. As concerning the ECC-based repair approach using separate CAMs, we use the fast computation of expression (2) to compute the size of the runtime-CAM and the diagnosis-CAM.

Concerning the ECC-based repair approach using separate CAMs and iterative diagnosis, using a diagnosis-CAM slightly larger than the runtime-CAM will require very large numbers of iterations, affecting test length adversely. Thus, we use diagnosis-CAMs several times larger than the runtime CAM. From proposition 2, the diagnosis is guaranteed to be successful if the number of *good* CAM locations is larger than the number of memory words affected by two or more faults. As the diagnosis-CAM is much larger than the runtime-CAM (which has sufficient size for repairing all memory words containing two or more faulty cells), the probability that the diagnosis-CAM comprises a number of *good* locations lower than the number of memory words containing two or more faulty cells is extremely low. Thus, the impact of the diagnosis-CAM on the yield will be extremely low, and we can use expression (1) instead of expression of (2). Then, the challenging issue concerning the iterative diagnosis approach concerns the computation of the number of iterations for any target size of the diagnosis-CAM. The automation of this task is addressed in the next section.

3.3.2 Test-length Computation

This section proposes algorithms for determining the increase of test length induced by the iterative diagnosis approach. This is a challenging task as the number of iterations of the test algorithm depends on the distribution of faults within the memory. This distribution concerns the distribution of faulty cells in the memory as well as the type of the fault affecting each faulty cell. The latter is necessary, as the number of iterations is determined by the number of errors produced during the execution of the test algorithm and their distribution. Thus, to determine the number of iterations for a fault distribution, we need to simulate the faulty memory together with the diagnosis circuitry for several iterations of the test algorithm (i.e. until an iteration of the test algorithm, which does not aborts the diagnosis of detected faults due to the saturation of the diagnosis-CAM). For large memories this is extremely time-consuming. In addition, to obtain statistically significant results, we need to repeat this process for a large number of times (up to 1000 times were used in our experiments). Furthermore, we evaluated the approach for 162 cases (3 different defect densities x 9 memory sizes x 6 diagnosis-CAM sizes). All these make the computation unfeasible. To address this challenge, we developed an innovative approach termed hereafter as pseudo-simulation (and a software platform implementing it), which provides the same result as conventional simulation but dramatically faster. This approach is described next.

3.3.2.1 Detection profiles

The distribution of the faults over the memory and their type, determine the positions within the test algorithm in which fault detections occur. Ultimately, the behavior of the iterative diagnosis scheme depends on these positions. Thus, to avoid performing fault injection experiments, and performing time consuming conventional fault simulation for each experiment, we adopted a much faster solution consisting in:

- Determining the detection profiles of the target set of faults for the target algorithm (i.e. the sequences of the test algorithm in which each fault of the target set is detected). For instance, if a fault is detected in sequences S1, S2, and S5 of a test algorithm, its detection profile is (S1; S2; S5).
- Associating the *conditional*[11] occurrence probability of each fault to its detection profile.
- Collapsing the obtained list of detection profiles and their probabilities by adding the probabilities of identical detection profiles. For instance, if two faults f1 and f2 have both the detection profile (S1; S3) and the *conditional* occurrence probabilities of f1 and f2 are P1 and P2, then we associate the probability P1+P2 to the detection profile (S1; S3).
- Performing detection-profiles injection experiments, and performing fast pseudo-simulation for each experiment.

For illustrating our approach let us consider a comprehensive set of fault models consisting in all static unlinked functional fault models (FFMs) involving one memory cell (single-cell FFMs) and two memory cells (two-cell FFMs) [29]. We also consider the march test algorithm MSS1 [31] shown in figure 1, which is an optimal test algorithm detecting all of them.

S(0): ⇳(W0); **S(1):** ⇑(R0, R0, W1, W1); **S(2)**: ⇑(R1, R1, W0, W0);
M01; M11; M12; M13; M14 M21; M22; M23; M24

S(3): ⇓(R0, R0, W1, W1); **S(4)**: ⇓(R1, R1, W0, W0); **S(5)**: ⇳(R0);
M31; M32; M33; M34 M41; M42; M43; M44 M51

Figure 1: MSS1 test algorithm

[11] The occurrence probability of each fault divided by the total probability that a fault of any kind occurs

To generate the detection profile for each fault, we need to identify which operations of the test algorithm detect it. Considering the target test algorithm, these operations can be generated manually, but this is a time consuming and error prone operation. Thus, using fault simulation is preferable because it is error-free and is very fast, as it requires simulating only one cell for each single-cell fault and only three cells for each two-cell fault (a victim cell, an aggressor cell with higher address than the victim cell, and an aggressor cell with lower address than the victim cell). Table 1 concerns the single-cell FFMs. The fourth column of this table presents the operations of MSS1 detecting each fault primitive, and the fifth column presents the list of sequences detecting each fault primitive. For instance, from the fourth column, the fault primitive < 0 / 1 / - > is detected by the operations (M11; M12) (M31; M32) and M51. Thus, the list of sequences in the fifth column is (S1; S3; S5). In the context of the iterative diagnosis approach, the detection sequences rather than the detection operations are of interest. Thus, in the following we are deriving sequence-based detection profiles. However, in applications where detection-operations are of importance (as for instance for determining the masking probability of the transparent BIST scheme proposed in section 6), the proposed pseudo-simulation approach can also be employed to reduce simulation time, but in this case we will use the detection operations shown in the fourth column of the table for creating the detection profiles. Using detection operations instead of detection sequences will result in a larger number of detection profiles. Hence, using sequence-based detection profiles, as allowed in the context of the present study, reduces computation time.

For the two-cell FFMs, the similar results are presented in table 2.

Note that, for all two-cell FFMs except the CFst <0; 0/1/- > and <1; 1/0/- >, the detecting sequence depends on the relation between the addresses of the victim and the aggressor cells. This is reported in the fifth column of table 2 as “half in Si the other half in Sj”.

Note also that, all FFMS are detected by MSS1 regardless to the state of the memory before the initializing sequence S0. However, depending on the state of each memory cell before S0, some faults can be additionally detected in S1. Such additional detections may saturate the diagnosis-CAM earlier and increase the diagnosis length. Thus, we also take into account such detections. Starting from the second iteration of the test algorithm the states of the memory cells before S0 are known (they are equal to their states at the last sequence of the test algorithm - the state 0 in MSS1 as determined from sequence S5). Thus, in tables 1 and 2 we have normally reported the detection operations and sequences as determined by these states. In addition at the end of these tables we have reported the detection operations and sequences concerning the first iteration of the test algorithm. As the state of each cell before S0 is unknown, we consider that each cell has 0.5 probability to be at the 0 state and another 0.5 probability to be in the 1 state before S0. For single-cell FFMS, this issue concerns TF <1w0/1/- > and WDF <0w0/↑/- >. Thus, in the last two rows of table 1 *we report again* the detection operations and sequences for these fault primitives, *considering this time* that before the sequence S0 of the first iteration of the test algorithm, the memory cells have 0.5 probability to be 1 and 0.5 probability to be 0. It results in the situations where certain detections have “50% chances”, as reported in the last two rows of the table. Note however that, as reported in the 4^{th} and 6^{th} rows of table 1, after the first iteration TF <1w0/1/- > gives the detection profile (S3; S5), and WDF <0w0/↑/- > gives the detection profile (S1; S3; S5). On the other hand, as reported in the last two rows of table 1, during the first iteration of the test algorithm TF <1w0/1/- > gives the detection profiles (S3; S5) and (S1; S3; S5), each with occurrence probability equal to the half of the occurrence probability of TF <1w0/1/- >. This is also the case for WDF <0w0/↑/- >. Then we find that, if the occurrence probabilities of the fault primitives TF <1w0/1/- > and WDF <0w0/↑/- > are equal, the occurrence probabilities of the detection profiles (S3; S5) and (S1; S3; S5) obtained from the fault primitives TF <1w0/1/- > and WDF <0w0/↑/- > are the same in all the iterations of the test algorithm. As for two-cell FFMS, the fault primitives concerned by these issue are CFdsxw!x <1w0; 0/↑/- > and CFdsxwx <0w0; 0/↑/- >. Thus, in table 2 we have reported their detection operations and sequences for both the first iteration and the subsequent ones. Again, if the fault primitives

CFdsxw!x <1w0; 0/↑/- > and CFdsxwx <0w0; 0/↑/- > have equal occurrence probabilities, we find that the detection profiles and their probabilities are the same in all iterations. The case where these probabilities are equal is easier to treat, as we treat identically all iterations. On the other hand, if these probabilities differ, we can apply our approach, but we need to create a specific detection profile for TF <1w0/1/- > differentiating the first iteration from the other iterations. That is: to all iterations except the first, we will associate the detection profile (S3; S5) with probability equal to the occurrence probability of TF <1w0/1/- >; while to the first iteration we will associate the detection profile (S1; S3; S5) with probability equal to the half of the occurrence probability of TF <1w0/1/- >, and the detection profile (S3; S5) with probability equal to the other half. We also treat similarly WDF <0w0/↑/- >, CFdsxw!x <1w0; 0/↑/- >, and CFdsxwx <0w0; 0/↑/- >.

Table 1: Detection operations and detection sequences for single-cell FFMs

#	FFM	Fault Primitives	Detected in operations:	Detection sequences
1	SF	< 0 / 1 / - >	(M11; M12); (M31; M32); M51	(S1; S3; S5)
		< 1 / 0 / - >	(M21; M22); (M41; M42)	(S2; S4)
2	TF	<1w0/1/- >	(M31; M32); M51	(S3; S5)
		<0w1/0/- >	(M21; M22); (M41; M42)	(S2; S4)
3	WDF	<0w0/↑/- >	(M31; M32); M51;	(S3; S5)
		<1w1/↓/- >	(M21 ; M22) ; (M41 ; M42)	(S2; S4)
4	RDF	< r0 / ↑ /1 >	(M11; M12); (M31; M32); M51	(S1; S3; S5)
		< r1 / ↓ /0 >	(M21; M22); (M41; M42)	(S2; S4)
5	DRDF	< r0 / ↑ /0 >	M12 ; M32	(S1 ; S3)
		< r1 / ↓ /1 >	M22; M42	(S2; S4)
6	IRF	< r0 / 0 / 1>	(M11; M12); (M31; M32); M51	(S1; S3; S5)
		< r1 / 1 / 0>	(M21; M22); (M41; M42)	(S2; S4)
2'	TF	<1w0/1/- >	(M31; M32); M51; 50% chances to be detected in (M11; M12).	(S3; S5), 50% in S1
3'	WDF	<0w0/↑/- >	(M31; M32); M51; 50% chances to be detected in (M11; M12).	(S3; S5), 50% in S1

Table 2: Detection operations and detection sequences for two-cell FFMs

#	FFM	Fault Primitive	Detected in operations:	# of detection sequences
1.a	CFst	<0; 0/1/- >	(M11; M12); (M31; M32); M51	(S1; S3; S5)
		<0; 1/0/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
1.b	CFst	<1; 0/1/- >	half in (M11; M12; M51); the other half in (M31; M32)..	(half in S1 and S5, the other half in S3)
		<1; 1/0/- >	M(21; M22); (M41; M42)	(S2 ; S4)
2.1.a	CFdsrx	<r0; 0/↑/- >	half in (M11; M12); the other half in (M31; M32)	(half in S1 the other half in S3)
		<r0; 1/↓ /- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4).
2.1.b	CFdsrx	<r1; 0/↑/- >	half in (M31; M32); the other half in M51	(half in S3 the other half in S5)
		<r1; 1/↓/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
2.2.a	CFdsxw!x	<0w1; 0/↑/- >	half in (M11; M12); the other half in (M31; M32)	(half in S1 the other half in S3)
		<0w1; 1/↓/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
2.2.b	CFdsxw!x	<1w0; 0/↑/- >	half in (M31; M32); the other half in M51.	(half in S3 the other half in S5)
		<1w0; 1/↓/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
2.3.a	CFdsxwx	<0w0; 0/↑/- >	half in (M31; M32); the other half in M51. All in (M11; M12).	(half in S3 the other half in S5)

		<0w0; 1/↓/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
2.3.b	CFdsxwx	<1w1; 0/↑/- >	half in (M11; M12); the other half in (M31; M32)	(half in S1 the other half in S3).
		<1w1; 1/↓/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
3.a	CFtr	<0; 1w0/1/- >	half in (M31; M32); the other half in M51	(half in S3 the other half in S5)
		<0; 0w1/0/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4).
3.b	CFtr	<1; 1w0/1/- >	half in (M31; M32); the other half in M51	(half in S3 the other half in S5)
		<1; 0w1/0/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
4.a	CFwd	<0; 0w0/↑/- >	half in (M31; M32); the other half in M51	(half in S3 the other half in S5)
		<0; 1w1/↓/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
4.b	CFwd	<1; 0w0/↑/- >	half in (M31; M32); the other half in M51	(half in S3 the other half in S5)
		<1; 1w1/↓/- >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
5.a	CFrd	<0; r0/↑/1 >	half in (M11; M12); the other half in (M31; M32). Also 100% in M51.	(half in S1 the other half in S3) plus all in S5
		<0; r1/↓/0 >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
5.b	CFrd	<1; r0/↑/1 >	half in (M11; M12); the other half in (M31; M32).	(half in S1 the other half in S3).
		<1; r1/↓/0 >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
6.a	CFdrd	<0; r0/↑/0 >	half in M12; the other half in M32	(half in S1 the other half in S3)
		<0; r1/↓/1 >	half in M22; the other half in M42	(half in S2 the other half in S4)
6.b	CFdrd	<1; r0/↑/0 >	half in M12; the other half in M32	(half in S1 the other half in S3)
		<1; r1/↓/1 >	half in M22; the other half in M42	(half in S2 the other half in S4)
7.a	CFir	<0; r0/0/1 >	half in (M11; M12); the other half in (M31; M32). Also always in M51.	(half in S1 the other half in S3) plus all in S5
		<0; r1/1/0 >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
7.b	CFir	<1; r0/0/1 >	half in (M11; M12); the other half in (M31; M32).	(half in S1 the other half in S3)
		<1; r1/1/0 >	half in (M21; M22); the other half in (M41; M42)	(half in S2 the other half in S4)
2.2.b'	CFdsxw!x	<1w0; 0/↑/- >	half in (M31; M32); the other half in M51. Also 50% chances to be detected in (M11; M12).	(half in S3 the other half in S5). 50% in S1
2.3.a'	CFdsxwx	<0w0; 0/↑/- >	half in (M31; M32); the other half in M51. Also 50% chances to be detected in (M11; M12).	(half in S3 the other half in S5). 50% in S1

To create the detection profiles with their occurrence probability we need to know the occurrence probabilities of the fault-primitives. As these probabilities are fab-sensitive data and are not available, to illustrate our approach we consider that all fault primitives have the same *conditional* occurrence probability. Thus, as we have 48 fault primitives, the *conditional* occurrence probability of each of them is taken as 1/48. Thus, the occurrence probability of each fault primitive is equal to Pf/48, where Pf is the probability that a cell is affected by a fault of any kind. Also, as mentioned earlier, for most of the two-cell FFMs the detecting sequence depends on the relation between the addresses of the victim and the aggressor

cells (this was reported in the fifth column of table 2 as "half in Si the other half in Sj"). Then, the occurrence probability of these faults has to be split in two parts, the one to be associated with Si and the other with Sj. This split may depend on the potential distribution of the aggressor cells in two groups: those having addresses that are lower than the address of the victim cell, and those having addresses that are higher than the address of the victim cell. One option is to consider that the number of victim cells of the first group is proportional to the number of memory cells that have lower addresses than the victim cell, and consider the similar for the second group. In this case, during fault injection, we will adapt the fault occurrence probability to the position of the cell in which we inject the fault. That is, for a memory bank comprising N words, there are @v - 1 addresses lower than the address @v of the victim cell, and N - @v addresses higher that the address of the victim cell. Thus, in this case the occurrence probability of the two-cell fault will be split in two parts (Pf/48)(@v - 1)/(N – 1) and (Pf/48)(N - @v)/(N – 1). However, it is more realistic to consider that the victim cells are distributed in the close proximity of the aggressor cell [37]. In this case we will have similar probabilities for the two groups. This approach was adopted in the case study considered here, but the different approach can also be implemented trivially.

Using the above considerations, we associate the corresponding probabilities to the different detection profiles defined by the detection-sequences instances reported in the fifth column of the tables 1 and 2. For example, as reported in the second row-fifth column of table 1 (detection sequence *(S1; S3; S5)*), the fault primitive < 0 / 1 / - > is always detected in the sequences S1, S3, and S5. Thus, we associate the probability Pf/48 of the fault primitive < 0 / 1 / - > to the detection profile (S1; S3; S5). On the other hand, as reported in the fourth row-fifth column of table 1 (detection sequence *(S3; S5), 50% in S1*), the primitive <1w0/1/- > is always detected in the sequences S3 and S5, and has 0.5 probability (50% chances in the table) to be detected in sequence S1. Thus, the Pf/48 occurrence probability of the primitive <1w0/1/- > is split in two detection profiles: Pf/96 for the detection profile (S3; S5), and Pf/96 for the detection profile (S1; S3; S5). Finally, as reported in the fourth row-fifth column of table 2 (detection sequence (*half in S1 and S5, the other half in S3)),* depending on the relation of the addresses of the aggressor cell and of victim cell, half of the faults corresponding to the primitive <1; 0/1/- > is always detected in S1 and S5, and the other half is always detected in S3. Thus, the Pf/48 occurrence probability of primitive <1; 0/1/- > is split in two detection profiles: Pf/96 for the detection profile (S1; S5), and Pf/96 for the detection profile (S3). This way, from tables 1 and 2 we obtain the detection profiles and their probabilities.
Then we get the identical detection profiles by adding their probabilities. The result is the 10 detection profiles reported in table 3, together with their occurrence probabilities.

Table 3: Statistical distribution of detection profiles.

Detection Profile	**Probability**
(S1)	Pf x 0.072916
(S2)	Pf x 0.17708333
(S3)	Pf x 0.14583333
(S4)	Pf x 0.17708333
(S5)	Pf x 0.0625
(S1; S3)	Pf x 0.03125
(S1; S5)	Pf x 0.04166666
(S2; S4)	Pf x 0.14583333
(S3; S5)	Pf x 0.04166666
(S1; S3; S5)	Pf x 0.10416667

3.3.2.2 Pseudo-simulation

To perform fault-injections over the memory we create a Nxm array DP[N, m] corresponding to the Nxm cells of the memory (N being the number of memory words and m the number of bits per word):

- The Nxm elements of this array (to be referred hereafter as cells) are visited one after the other.
- For each cell the rand function of MATLAB is used to generate discrete event sampling for the 10 detection profiles of the first column 1 of Table 3, following their probabilities given in the second column of the table.
- The resulting event (i.e. one of the 10 *detection profiles* or the *empty detection profile* corresponding to a fault-free cell) is written in the cell.
- Then another cell is visited and so on …

Each time we execute an injection experiment as described above, we need to determine the number of test iterations required to diagnose the faulty memory described by the array DP[N, m]. Using conventional memory-fault simulation this is extremely time consuming. Fortunately, our detection-profiles-based approach avoids simulating the faults injected in the memory during the execution of the test algorithm. It also avoids the simulation of the diagnosis hardware, and more particular of the diagnosis-CAM, which is very time consuming. Indeed:

i. As during the fault injection process we stored in each element of DP[N, m] the test sequences in which the injected fault is detected (detection profile), we do not need to perform the time-consuming task of conventional fault simulation during the execution of the test algorithm.
ii. In the real diagnosis circuit, we save in a CAM the address of each faulty cell. Simulating algorithmically the operation of a CAM requires time-consuming searches. In our case, we are able to obtain the expected result by using the detection profiles to count the number of occupied CAM locations, reducing drastically the computation time.
iii. In the real diagnosis circuit, after an *aborted Miss-updating* we need to execute the algorithm until its end, in order to check if after the *aborted Miss-updating* the test algorithm detects faulty cells, which belong to faulty words that were detected by the test algorithm and stored in the CAM before the *aborted Miss-updating*. As each row of DP[N, m] stores all faults affecting the corresponding memory word, we do not need to execute the test algorithm until its end for checking this characteristic. Thus, in the pseudo-simulation, each iteration of the test algorithm ends when an *aborted Miss-updating* occurs.
iv. As the sensitization of the faults was already considered during the creation of the detection profiles. Then, in the pseudo-simulation, after an *aborted Miss-updating* occurring during an iteration I we can start the next iteration $I+1$ of the test algorithm from the test operation in which the *aborted Miss-updating* has occurred.

From the points i and ii, the proposed approach avoids the very time-consuming conventional fault simulation. It also avoids the algorithmic simulation of CAM searches, which is less time consuming than the fault simulation, but it is also infeasible in reasonable computation time[12]. In addition from points iii

[12] Each time a memory address is visited by the iterative diagnosis process it has to be compared serially with all CAM contents. This results in $N_w x N_{cw} x N_{ms} x N_i x N_{fi}$, where Nw is the number of memory words, N_{cw} is the number of CAM words, N_{ms} of march sequences of the test algorithm, N_i is the number of iterations of the diagnosis algorithm, and N_{fi} is the number of fault injections required to reach statistical significance. Thus, for a memory comprising $250x10^6$ words, a defect density $N_f = 10^{-3}$ (which will require a full diagnosis CAM of about $10x10^6$ words), a reduced CAM having size equal to the one fourth of the full diagnosis-CAM (i.e. $2.5x10^6$ words), the MSS1 test algorithm (which employs 5 march elements), a diagnosis process requiring 7

and iv, the simulation length is reduced significantly. Indeed, as at each iteration] the pseudo simulation ends the execution of the test algorithm at the test operation during which an *aborted Miss-updating* occurs, and at the next iteration]+1 starts the execution of the test algorithm from the same test operation, then, none of the operations of the test algorithm is executed twice except the operations corresponding to an *aborted Miss-updating*. Thus, if the length of the test algorithm is *tl* and the number of iterations required by the diagnosis process is *ni*, the length of the pseudo-simulation will be only *ni* + *tl* – 1 instead of *ni*x*tl*, resulting in non-negligible time reduction. The number of iterations depends on the diagnosis-CAM size and the number of faulty words. Using small diagnosis-CAM to reduce cost increases the number of iterations and the length of the conventional simulation. This also happens for constant CAM size and increasing defect densities. However, the length of the pseudo-simulation is roughly constant (equal to the test algorithm length plus one extra operation per test algorithm iteration). Thus, the execution time remains low in all situations.

To simplify the pseudo-simulation we perform a pretreatment of the DP[N, m] array, to create a NxS array BS[N, S], where S = q + 1, and q is the number of sequences of the test algorithm (e.g. for MSS1 we will obtain a Nx6 array BS[N, 6]). The elements of each row MA of BS[N, S] are S Boolean variables: BS1(MA), BS2(MA), … BSq(MA), and BM(MA) computed from the elements of the row MA of DP(N, m) by means of the following process:

BS-create

- BS1(MA) is set to 1 if the detection profile of some cell in the row MA of DP[N, m] includes S1. BS2(MA), BS2(MA), … BSq(MA), are set similarly.
- BM(MA) is set to 1 if more than one of the cells in row MA have nonempty detection profile (i.e. BM(MA) = 1 means that the memory word corresponding to row MA contains two or more faulty cells).

In the pseudo-simulation algorithm, we also need to know the number of *good* CAM locations. From section 3.1.1, the probability that a CAM location is good is given by $Pwcg = (1 - P_f)^{rN@+qNf+N}$. We create a N_{WDC}x1 array (where N_{WDC} is the number of the locations of the diagnosis-CAM). We initialize all elements of this array to 0 and for each element of this array we perform fault injection with probability equal Pwcg. If the outcome of the fault injection is the faulty-state we write 1 in the current element of the array. Then we count the elements of the array containing 0, and *we allocate the outcome k to the integer variable cg* (representing the number of *good* locations of the diagnosis-CAM).

Next we present the pseudo-simulation algorithm, but instead of presenting an algorithm dedicated to MSS1, we present a generic algorithm because:

- It can be used with any memory test algorithm.
- It is more compact as it comprises a single generic subroutine valid for any test sequence, instead of a 5 distinct subroutines corresponding to the 5 test sequences of MSS1.
- For any iteration starting updating the CAM in sequence S(i), a Boolean condition c(i) has to be computed to indicate that a new CAM location is occupied. These Boolean conditions become very complex as i increases and their manual generation is very time consuming and prone to errors. For instance, for an iteration starting updating the CAM in sequence S(2) of MSS1 we find:

c(2) = ((PS(ma,2)==1) & ((PS(ma,1) == 0) or ((PS(ma,6)==0) & (((sr==1) & (ma < ar)) or (sr==2)))))

This increases quickly with i, and for i=3, i = 4, and i = 5, we find:
c(3) = ((PS(ma,3)==1) & (((PS(ma,1) == 0) & (PS(ma,2) == 0)) or ((PS(ma,6)==0) & (((PS(ma,1) == 1) & (PS(ma,2) == 0) & (((sr==1) & (ma < ar)) or (sr==2) or (sr==3))) or ((PS(ma,2) == 1) & (((sr==2) & (ma < ar)) or (sr==3)))))))

---

iterations, and 1000 fault injections (as used in our experiments to guaranty statistically significant results), the number of comparisons required for the algorithmic simulation of the CAM will be $250x10^6x2.5x10^6x5x7x1000 = 2.2x10^{19}$ comparisons!

c(4) = ((PS(ma,4)==1) & (((PS(ma,1) == 0) & (PS(ma,2) == 0) & (PS(ma,3) == 0)) or ((PS(ma,6)==0) & (((PS(ma,1) == 1) & (PS(ma,2) == 0) & (PS(ma,3) == 0) & (((sr==1) & (ma < ar)) or (sr==2) or (sr==3) or (sr==4))) or ((PS(ma,2) == 1) & (PS(ma,3) == 0) & (((sr==2) & (ma < ar)) or (sr==3) or (sr==4))) or ((PS(ma,3) == 1) & (((sr==3) & (ma > ar)) or (sr==4)))))))

c(5) = ((PS(ma,5)==1) & (((PS(ma,1) == 0) & (PS(ma,2) == 0) & (PS(ma,3) == 0) & (PS(ma,4) == 0)) or ((PS(ma,6)==0) & (((PS(ma,1) == 1) & (PS(ma,2) == 0) & (PS(ma,3) == 0) & (PS(ma,4) == 0) & (((sr==1) & (ma < ar)) or (sr==2) or (sr==3) or (sr==4) or (sr==5))) or ((PS(ma,2) == 1) & (PS(ma,3) == 0) & (PS(ma,4) == 0) & (((sr==2) & (ma < ar)) or (sr==3) or (sr==4) or (sr==5))) or ((PS(ma,3) == 1) & (PS(ma,4) == 0) & (((sr==3) & (ma > ar)) or (sr==4) or (sr==5))) or ((PS(ma,4) == 1) & (((sr==4) & (ma > ar)) or (sr==5)))))))

The generic pseudo-simulation algorithm is presented below. It uses the Boolean function c(i), and another Boolean function V(i), computed by dedicated subroutines presented later.
Note also that, for more intuitive reading, the symbol "|" of the logic OR operator is replaced by "or".

```
% Generic simulation algorithm for any march test composed
% of q test sequences S(1), S(2), … S(q)
S = q + 1; r = q; cg = k;
ni = 1; dl = 0;          % ni and dl will count respectively the number of iterations of the diagnosis
                         % process and its number of operations (diagnosis length);
co = 0; cmo = 0;         % co; and cmo; will count respectively the number of occupied CAM locations
                         % and the number of CAM locations occupied by words containing one or more
                         % faulty cells;
sr = 0; ar = 0;          % sr and ar indicate respectively the sequence and the memory address
                         % at which the latest aborted Miss-updating has occurred.
PS[N,S] = BS[N,S];       % BS[N,S] is generated by process BS-create
for i = 1 : q
   ud(i) = si;           % si = 1 if test sequence S(i) uses up addressing, si = -1 otherwise.
   tl(i) = tl – rl(i)    % tl is the length of the test algorithm and rl(i) is the reduction of the test-algorithm for
                         % an iteration starting updating the CAM at sequence S(i), as allowed by corollary 1
                         % rl(1) = 0; rl(2) = N; rl(3) = 4N; rl(4) = 9N; rl(5) = 12N
end
for i = 1 : r
   % Simulation of any test sequence S(i)
   if (ud(i) == 1)
      for ma = 1 : N
         if c(i)
            co = co + 1;
         end
         if co == cg+1
            co = cmo +1; sr = i; ar = ma; dl = dl + tl(i); ni = ni +1;
         end
         if ((PS(ma,S) == 1) & (PS(ma, i) == 1) & (V(i)))
            cmo = cmo + 1;
         end
      end
   else
```

```
        for ma = N : (-1) : 1
            if c(i)
                co = co + 1;
            end
            if co == cg+1
                co = cmo +1; sr = i; ar = ma; dl = dl + tl(i); ni = ni +1;
            end
            if ((PS(ma,S) == 1) & (PS(ma, i) == 1) & (V(i)))
                cmo = cmo + 1;
            end
        end
    end
end
```

% Computation of condition c(i)

```
c(i) = ((PS(ma,i)==1) &(V(i) or ((PS(ma,S)==0) & W(i))))
```

% Computation of V(i)

```
V(i) = 1;
for j = 1:i-1
    V(i) = (V(i) & (PS(ma, j) == 0));
end
```

% Computation of W(i)

```
W(i) = 0;
for p = 1:i-1
    if  (ud(p) == 1)
            W(i)=    (W(i) or ((PS(ma,p) == 1) & T(p,i) & (((sr==p)
            & (ma < ar)) or U(p,i) )) );
    else (ud(p) = -1)
            W(i)=    (W(i) or ((PS(ma,p) == 1) & T(p,i) & (((sr==p)
            & (ma > ar))or U(p,i) )) );
    end
end
```

% Computation of T(p,i)

```
T(p,i) = 1;
for j = (p+1):i-1
    T(p,i) = T(p,i) & (PS(ma,j) == 0);
end
```

% Computation of U(p,i)

```
U(p,i) = 0;
for j = p+1:i
    U(p,i) = U(p,i) or (sr==j);
end
```

3.4 EVALUATION

The size of the CAMs required for achieving 90% yield for the three repair schemes, is estimated by the analytical expressions presented in chapter 5. The area, power, and performance of the embedded SRAMs and of the CAMs are estimated by using CACTI [[34] for 45nm process. The mean number of test iterations

was evaluated by the approach presented in sub-section 3.3.2. This approach is very fast compared with conventional fault simulation and allowed us performing intensive statistical fault injection experiments (1000 fault injection campaigns and simulations for each SRAM case, defect density and diagnosis-CAM size). The results are presented in Table 4.

Column 1 in Table 4 gives the fault occurrence probabilities P_f that we experimented. All cases presented in this table concern a total of 9,75 Gbit SRAM capacity embedded in a SoC, corresponding to a total of 250M words x 39 bits per word (32 data bits and 7 Hamming code bits). The 9,75 Gbit total memory capacity is split in numerous embedded memories distributed over the SoC. We have considered 3 cases for this distribution: 300 embedded memories; 1000 embedded memories; and 3000 embedded memories, as reported in Column 2. Columns 3 and 4 show the area and power penalties (as a percentage of area and power of the memory under repair) for conventional (non-ECC) repair. Column 5 gives the runtime power penalty for all cases of ECC-based repair. In fact, in all these cases we use separate runtime-CAM, which stores the words comprising multiple faults. Thus, the runtime-CAM is the same for all these cases, resulting in the same runtime power. Column 6 gives the area penalty for the approach using full-size diagnosis-CAM and separate runtime-CAM. Columns 7 and 8 give the area penalty and the *mean* number of test iterations, when we use a diagnosis-CAM having the half size of the full-size diagnosis-CAM. Columns 9 and 10 give the same metrics when the diagnosis-CAM is one fourth of the full-size diagnosis-CAM. Columns 11 and 12 give the same metrics when diagnosis-CAM is one sixth of the full-size diagnosis-CAM.

In Table 4 we observe that: The area and power penalties for the conventional repair (referred as *Appr*. 1 in the Table) grows roughly linearly with the defect density. However, the area penalty is significant but not huge while the power penalty becomes excessive.

The first of our proposed approaches (ECC-based repair using separate runtime-CAM and full-size diagnosis-CAM – *Appr. 2*), allows dramatic reduction of runtime power, which now increases sub-linearly with respect to the defect density. This approach does not affect test time, but its area penalty is similar to the conventional repair scheme.

Table 4. Results.

P_f	#Emb Mem	**Appr. 1** Non-ECC Repair		All ECC Rep.	**Appr. 2** ECC-Rep Separate CAM	**Appr. 3** ECC-Rep. CAM/2		**Appr. 4** ECC-Rep CAM/4		**Appr. 5** ECC-Rep CAM/6	
		%A	%P	%P	%A	%A	#It	%A	#It	%A	#It
10^{-4}	300	1.317	185.4	1.267	1.326	0.696	3.00	0.313	6.00	0.215	9.00
	1000	1.195	96.06	1.185	1.211	0.619	3.00	0.330	6.05	0.236	9.01
	3000	1.237	67.90	1.297	1.265	0.675	2.89	0.382	5.26	0.279	7.73
$3x\ 10^{-4}$	300	3.933	533.0	5.337	3.969	2.158	3.00	1.029	6.00	0.703	9.00
	1000	3.873	283.3	4.431	3.936	2.054	3.00	0.939	6.10	0.647	9.19
	3000	3.459	177.9	3.676	3.537	1.818	3.00	0.967	5.90	0.679	8.39
10^{-3}	300	12.75	1629	39.56	13.00	6.836	3.00	3.695	7.00	2.65	10.00
	1000	13.07	913.9	24.18	13.36	7.337	3.00	3.586	6.88	2.50	10.00
	3000	13.50	581.5	17.56	13.84	6.505	3.00	3.146	6.01	2.24	9.16

The second set of our proposed approaches (ECC-based repair using separate runtime-CAM and reduced diagnosis-CAM – *Appr. 3, 4, and 5*), achieves the same runtime power reduction as the first approach. In

addition, it allows significant area penalty reduction (especially for the higher defect densities). This benefit comes at the expense of extra test length, since the test algorithm will have to be executed about 3 times for the CAM/2 case, 5 to 7 times for CAM/4, and 8 to 12 times for the CAM/6 case, as indicated in the #It columns computed by our statistical injection and simulation tool. Note however that, these approaches use much smaller diagnosis-CAM with respect to the *non-ECC repair* approaches (Appr. 1) and the approach using a full-length separate CAM (Appr. 2). Thus, they will have lower power dissipation during the test and diagnosis phase. This fact can be exploited to test in parallel a larger number of memories and reduce the impact on test time. Then, as shown in table 5, the test time increase is much smaller than the test length increase. Furthermore, as the area penalty increases linearly with the defect density and embedded memories occupy the largest part of modern SoCs (more than 90% of the SoC area in most cases), this penalty becomes totally undesirable. For instance, for the 10^{-3} defect density the area penalty is about 13%, which in a SoC in which embedded memories occupy more 90% results in an area penalty exceeding 11.7% of the total SoC area. Thus, the area penalty reduction achieved by the iterative diagnosis approach is highly desirable.

Table 5. Test-time increase

P_f	#Embedded Memories	Conventional Repair	**Appr. 3** ECC-Rep. CAM/2		**Appr. 4** ECC-Rep. CAM/4		**Appr. 5** ECC-Rep. CAM/6	
		Test Power	Test Power	Test-time increase	Test Power	Test-time increase	Test Power	Test-time increase
10^{-4}	300	2.85	1.99	**2.01**	1.47	**3.09**	1.33	**4.20**
	1000	1.96	1.51	**2.31**	1.27	**3.92**	1.19	**5.47**
	3000	1.68	1.36	**2.34**	1.18	**3.69**	1.13	**5.20**
$3x10^{-4}$	300	6.33	3.92	**1.86**	2.43	**2.30**	1.99	**2.83**
	1000	3.83	2.53	**1.98**	1.71	**2.72**	1.49	**3.57**
	3000	2.78	1.93	**2.01**	1.49	**3.16**	1.33	**4.01**
10^{-3}	300	17.29	9.74	**1.69**	5.69	**2.30**	4.32	**2.50**
	1000	10.14	6.01	**1.78**	3.44	**2.33**	2.69	**2.65**
	3000	6.81	4.07	**1.79**	2.47	**2.18**	2.01	**2.70**

These techniques, complement the approach proposed in [35], which completely eliminates the diagnosis-CAM at the expense of significant increase in test time. They, result in a comprehensive framework enabling:

- Repairing memories affected by high defect densities at low area and power costs;
- Trading area penalty with test time.
- Dealing at the same time with high defect densities and aggressive voltage reduction, to aggressively reduce power.

3.5 CONCLUSION

In chapter 2 we proposed and developed a new family of memory test algorithms (SRDF test algorithms), which eliminate the diagnosis circuit in ECC-based memory repair. In high defect densities it results in dramatic reduction of the area and power penalties with respect to conventional memory repair. However, the length of SRDF test algorithms is increased significantly with respect to conventional test algorithms.

To reduce this length, in this chapter we proposed two solutions. The first, instead of employing a single CAM for both diagnosis and repair purposes (as do existing repair schemes), it employs a diagnosis-CAM used during the test and diagnosis phase, and a repair CAM used at runtime. Thus, this scheme reduces

drastically runtime power and also uses conventional test algorithms. However, area penalty may become very high, as the size of the diagnosis CAM is the same as the size of the CAM used in conventional memory repair. As memories occupy a very large amount of the SoC area (usually more than 90%), this high area penalty is very penalizing, since it will represent a high percentage of the total area of the SoC).

Thus, in this chapter we also propose an iterative diagnosis scheme, which reduces the size of the diagnosis CAM and compensates the reduced CAM space by: executing the test algorithm multiple times, diagnosing at each iteration a subset of the faulty memory words, and emptying CAM space at the end of this iteration to provide space for treating new faults at the next iteration. The first challenge of this approach is: how to ignore during an iteration of the test algorithm faulty words eliminated from the CAM during the earlier iterations, as this elimination erases all information concerning them. Due to this lack of information, this elimination can result in fault-masking for some faults and in miss-diagnosis of words containing multiple faulty cells. This issue was resolved by a dedicated iterative diagnosis algorithm, which is formally shown to correctly diagnose all words containing more than one faulty cell. The second challenge concerns the evaluation of this approach. This evaluation requires performing large numbers of fault injections, in order to obtain statistically significant results, and simulating for each of these injections the faulty memory and the diagnosis circuit by executing the diagnosis algorithm over them. As fault simulation is a very time-consuming process (and this is also the case for the algorithmic simulation of CAMs), we have developed a pseudo-simulation approach, which reduces dramatically simulation time while providing identical results as conventional fault simulation.

The approaches proposed in this chapter offer a large space for trade-offs in terms of area penalty and test time. Thus, in combination with the approach developed in chapter 2 for completely eliminating the diagnosis CAM, and the approach developed in chapter 5 enabling drastic runtime power reduction, they offer a comprehensive framework enabling making the most appropriate trade-offs in terms of area, power and test time to fit the constraints of each particular design.

CHAPTER 4

Low-Power Memory Repair Arcitectures

In the previous chapters we have shown that the advantages of ECC-based repair can be lost due to diagnosis issues, and we proposed and developed innovative solutions coping with this issue by one of the following means: using a new family of test algorithms; or using separate diagnosis-CAM and repair-CAM; or employing an iterative diagnosis process. These solutions make practical the ECC-based repair approach, which, for high-defect densities, reduces dramatically area and power cost with respect to conventional repair. However, although power-dissipation is reduced dramatically with respect to conventional repair, it remains significant as defect densities become very high. This is undesirable due to stringent low-power constraints in advanced technologies. To cope with this issue, in this chapter we propose and develop a new word repair architecture enabling drastic power-dissipation reduction. These architectures require new analytical expressions for evaluating their yield. However, the computational complexity of these expressions is very high even if we exploit the iterative relations that we developed for the yield computation of the repair schemes developed in chapters 1 and 2. To cope with this issue, dedicated yield computation mathematics was developed in chapter 5 and was used in this chapter for evaluating the new repair architecture.

4.1 PARTITIONING-BASED MEMORY REPAIR

In conventional word repair the addresses of faulty memory words are stored in the tag fields of a CAM [2][4]. Then, at each read or write operation, the memory address is compared with all tag-fields of the CAM. In case of hit, the operation is performed over the CAM, while, in case of miss, the operation is performed over the memory. The same word-repair architecture is used in ECC-based repair, with the difference that the CAM stores only the words comprising two or more faulty cells. In high defect densities, this reduces dramatically the size of the CAM, and consequently the related area and power penalties. However, as the CAM is power hungry (during each memory operation it has to compare the current address against the addresses of all memory words stored in the tag fields of the CAM), for high defect densities the power dissipation is still high. Thus, in this section we propose new repair architectures avoiding comparing the memory address against all the tag fields that store faulty memory addresses.

4.1.1 Cache-Based Repair

To reduce the power dissipation we need to reduce the number of tag comparisons. This can be done by partitioning the memory words into several sets corresponding to the possible combinations of a subset of address bits. Thus, by selecting d address bits we partition the memory words into 2^d sets, each

corresponding to one of the 2^d value combinations of these bits. Then, we can associate a CAM to each of these sets, and use each of these CAMs to repair words belonging to the corresponding set of memory words. As for each value of the d address bits the potential faulty memory words are in a given CAM, we can decode these bits and select one of these CAMs at a time. Thus, comparisons are performed in one CAM at a time, reducing the related power dissipation.

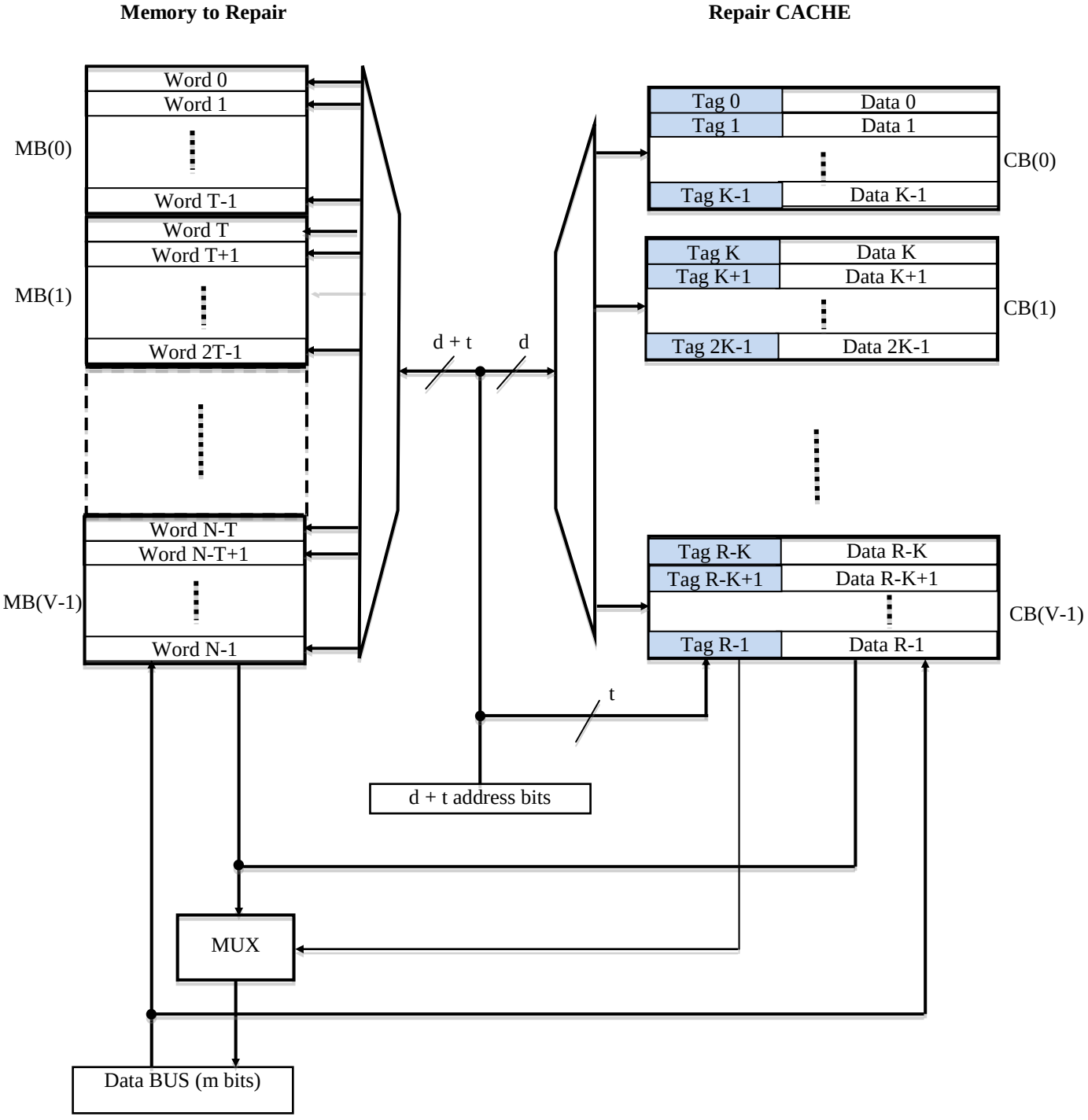


Figure 1. Cache-based repair architecture for low-power dissipation

We remark that the proposed partitioned-CAM scheme corresponds to a set associative cache. Thus, as shown in figure 1, the proposed repair architecture uses a repair cache instead of a repair CAM. In this figure, the memory under repair has n address bits addressing N words of m bits each. The n address bits are partitioned into two sets of d and t bits (i.e. n = d + t). The set-associative cache has $V = 2^d$ sets. Conventional address decoding is used to select at each memory access a set of the cache, by decoding the current value of the d address bits of the memory. Each cache set has K locations known also as ways (K-ways set-associative cache). Thus, the cache has a total of VxK locations. So, the cache in figure 1 consists in V physical blocks CB0, CB1, … CB(V-1), which implement the V sets. Each of these blocks comprises K locations (the K-ways). Each location comprises a tag field, a data field and at least two flag bits (the valid bit indicating if the location contains valid data, and the fault-free bit indicating if the CAM location is fault-free) [19]. These bits are not shown in figure 1.

When the d address bits take a given value, a memory word is selected according to the value of the remaining t address bits. As the t address bits can take $T=2^t$ values, there are T memory words associated to each value of the d address bits. Thus, the d address bits create a virtual partition of the memory into $V = 2^d$ blocks MB(0), MB(1), … MB(V-1). Each of these blocks contains T memory words. Thus, faulty words belonging to a virtual memory block (defined by a given value of the d address bits), will be repaired by the K locations of the cache block selected by this value of the d address bits. Therefore, each virtual memory block (e.g. MB(i)) can be viewed as a memory which has t address bits, comprises T words of m bits, and is repaired by a CAM having K words (block CB(i)). That is, MB(0) is repaired by CB(0), MB(1) by CB(1), … MB(V-1) by CB(V-1). Hence, we can consider that we have V memories of N/V words each, and each of them is repaired by a CAM of K words.

Therefore, test and diagnosis are performed as in CAM-based repair, with the specificity that the t address bits of a faulty word are stored in the tag field of the cache set selected by the d bits of the faulty address (while in CAM repair faulty addresses can be stored anywhere in the CAM).

Also, run-time operation is similar to CAM-based repair. That is, each read and write operation is performed over both the memory and the cache. Furthermore, during read, the hit signal determines which data are transferred to the data BUS. Thus, in figure 1, for hit = 0 the MUX transfers to the BUS the read data coming from the memory, while for hit = 1 it transfers the read data coming from the cache.

Concerning yield estimation, from the above discussion it is also obvious that to compute the yield of a memory repaired by means of a set-associative cache, we can employ the expression computing the yield for V memories repaired by using CAM repair.

This discussion also highlights the weak point of cache-based repair. In CAM repair, any CAM word can repair any faulty memory word, but in cache repair the words of each set can repair only memory words belonging to the corresponding virtual memory block. Thus, a memory in which the number of faulty words is lower than the total number of good locations of the cache, may not be repaired if a virtual memory block MB(i) comprises more faulty words than the *good* locations of the cache set CB(i). On the other hand, in CAM repair, if the CAM comprises more *good* locations than the faulty memory words, the repair will succeed regardless to the distribution of the faulty words within the memory. Thus, cache-based repair requires larger number of cache locations for achieving a given yield. This increase becomes higher if we try to reduce the number of ways of each set of the repair cache (by reducing the size of the virtual-memory blocks, and increasing their number as well as the number of sets of the cache), in order to reduce the power dissipation of the repair cache. An architecture coping with this additional constraint is presented next.

4.1.2 Overflow Repair Architecture

As discussed previously, trying to reduce power by increasing the number of virtual memory blocks and decreasing their size has unfavorable impact on the cache size. Indeed, decreasing the number of words of each virtual memory decreases the population of the words repaired by each cache set. As population

reduction increases the standard deviation of a statistical distribution, we will observe an increase of the deviation of the number of faulty memory words affecting the virtual memory blocks. This increase, combined with the increase of the number of virtual memory blocks, will increase the probability that for some block the number of faulty words is within the right-hand tail of the distribution curve. In other words, the probability that some virtual memory block contains a number of faulty words that is much larger than the mean value of faulty words affecting a virtual memory block is increased. Then, as the memory is repaired only if all virtual blocks are repaired, we will need for each cache set a number of locations much larger than the mean value of memory words affecting a virtual block. As a result, trying to reduce the power dissipation by increasing the number of cache sets may adversely impact the total number of cache locations, and therefore area and at certain extend power.

Because this issue occurs when the number of faulty memory words of a virtual block is in the tail of their distribution, it will affect a small number of sets. Thus, the total number of unrepaired memory words will be moderate. Hence, we can cope with this issue by adding an extra CAM of moderate size to repair these words. Therefore, we extend the architecture proposed in section 4.1.1 by adding a CAM referred as Overflow CAM in figure 2. This CAM will be used in the following manner.

During test and diagnosis a faulty memory address is allocated to the Overflow CAM only when the cache set that could repair this word is saturated.

At runtime, each read and write operation is performed in parallel over the memory, the cache and the Overflow CAM. Furthermore, as shown in figure 2, the MUX and its control logic driven by the hit signals of the cache and of the Overflow CAM, determine which read data are transferred to the data BUS. When the hit signal of the Overflow CAM is high, the MUX transfers to the BUS the read data coming from the Overflow CAM. When the hit signal of the Overflow CAM is low and the hit signal of the cache is high, the MUX transfers to the BUS the read data coming from the cache. When both hit signals are low, the MUX transfers to the BUS the read data coming from the memory.

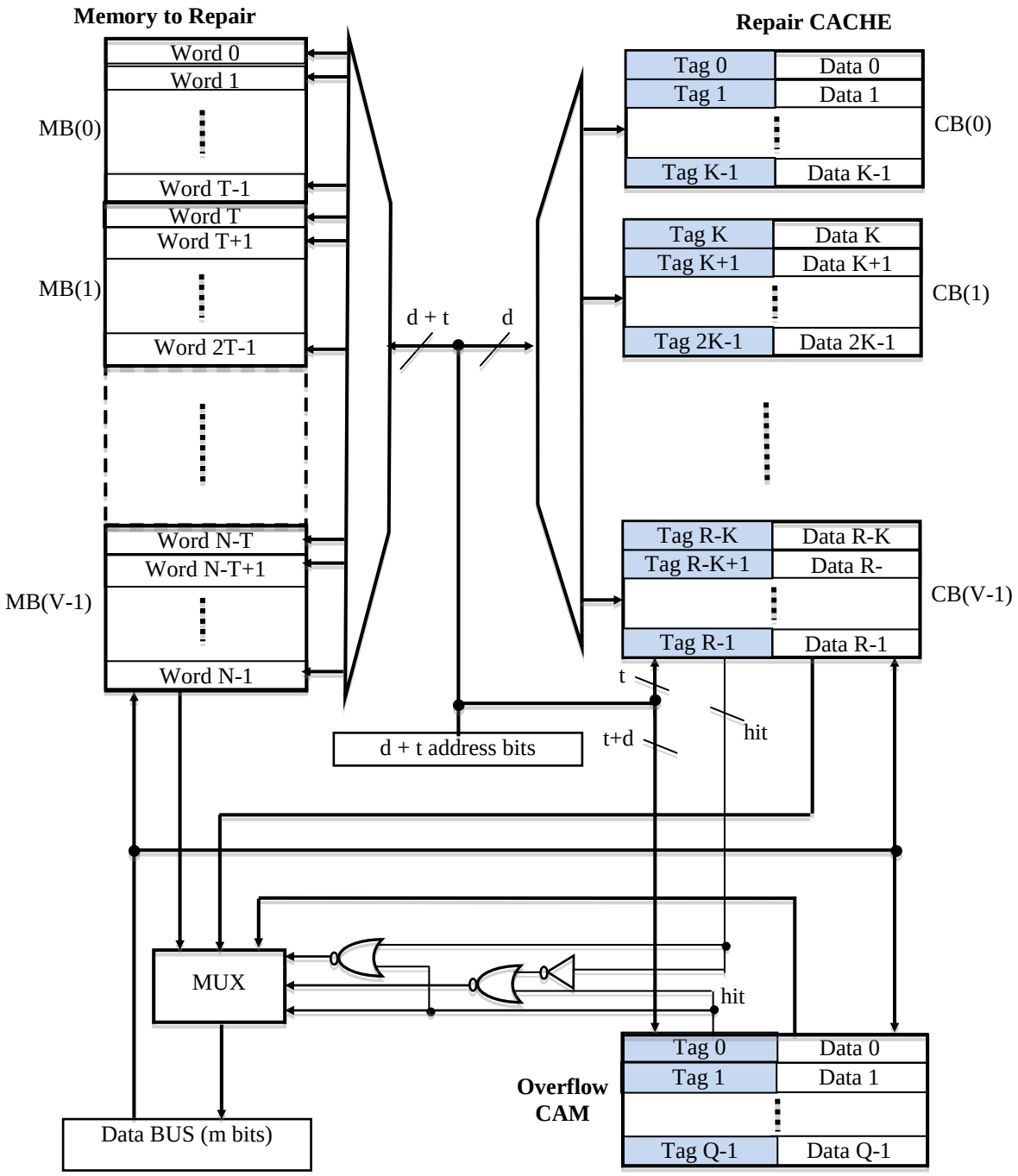


Figure 2. Overflow CAM repair architecture

The Overflow CAM can repair memory words left unrepaired by the cache regardless to their position within the memory. However, the CAM is power hungry, as it compares the memory address of the current read or write operation against all tags stored in the CAM. Thus, we have interest to maintain low the number of these comparisons. For doing so, we can replace the Overflow CAM by an Overflow Set-Associative Cache. To avoid confusing the Overflow Set-Associative Cache with the set-associative cache of figure 2, let us call the latter as CACHE 1 and the former as CACHE 2. The organization and operation of the Overflow Set-Associative Cache (CACHE 2) is quite similar to the set-associative cache (CACHE 1)

of figure 2. One difference with respect to CACHE 1 is that, similarly to the Overflow CAM, during test and diagnosis a faulty word is stored in CACHE 2 only in case of saturation of the set of CACHE 1 in which the faulty word could be stored. Also, the hit signal of CACHE 2 is used in the similar manner as the hit signal of the Overflow CAM.

4.2 YIELD COMPUTATION

In our experiments we will have to compute the yield for a set of memories. This is because we may want to consider systems comprising several memories, but also because, in the partitioning-based memory repair architectures proposed in the previous section, the yield computation for a single memory will be done by considering that it is composed of several smaller memories. The yield of system of M memories is, $Y_{SYS} = Y_1 Y_2 \ldots Y_M$ with $Y_{SYS} = Y^M$ when the M memories are identical.

To evaluate all architectures discussed so far, we need to dispose yield computation approaches for two architectures:

- The CAM-based repair architecture. This will also allow computing the yield for cache-based repair. Indeed, in this case we can consider that we have a system of M memories, where M is the number of virtual memory blocks. Thus, we can compute the yield Y for each virtual memory block by considering it as a memory repaired by a CAM having a number of locations equal to the number K of locations of each set of the set-associative cache. Then, the global yield of the memory will be given by $Y_{MEM} = Y^M$.
- The Overflow CAM repair architecture. This will also allow computing the yield for the repair architecture using the Overflow Set-Associative Cache (CACHE 2). Indeed, let N be the number of words of the memory under repair, V1 and K1 be respectively the numbers of sets and of ways of CACHE 1, and V2 and K2 be respectively the number of sets and of ways of the Overflow Set-Associative Cache (CACHE 2). Then, the yield computation for this repair architecture can be done by considering that we have a system of V2 memories such that: each of the V2 memories comprises N/V2 memory words, and is repaired by means of a set-associative cache having V1/V2 sets of K1 ways each, and an Overflow CAM of K2 locations. Thus, we can use the yield computation approach for the Overflow CAM repair architecture to compute the yield Y of each of the V2 memories. Then, the yield of the memory will be given by $Y_{MEM} = Y^{V2}$.

4.2.1 Yield Computation for Overflow CAM Repair

The yield for CAM-based repair can be computed by means of the following expression used in the yield evaluations of chapters 1 and 2 and derived in chapter 5, where N_{WM} and N_{WC} are respectively the number of memory words and the number of CAM words; and P_{WMG} and P_{WCG} are respectively the probability of a memory word to be good and the probability of a CAM location to be good.

$$Y = \sum_{t=0}^{N_{WC}} \left(\frac{N_{WM}! P_{WMG}^{(N_{WM}-t)}}{(N_{WM}-t)!t!} (1 - P_{WMG})^t \sum_{r=0}^{N_{WC}-t} \frac{N_{WC}! P_{WCG}^{(N_{WC}-r)}}{(N_{WC}-r)!r!} (1 - P_{WCG})^r \right) \quad \textbf{(1)}$$

As we have seen earlier, this expression can also be used to compute the yield of the set-associative cache repair architecture.

To compute the yield of the Overflow CAM repair architecture we need to develop a new analytical approach. Then, as discussed earlier, the same approach can be used to compute the yield for the Overflow Set-Associative Cache repair architecture. Thus, bellow we shortly discuss the yield computation for the Overflow CAM Repair architecture (its detailed derivation is presented in chapter 5).

Let N_S be the number of sets (Set(1), Set(2), … Set(N_S)) of the set-associative cache. As we have seen in section 4.1, the memory is virtually partitioned into N_S memory blocks MB(1), MB(2), … MB(N_S) repaired

respectively by Set(1), Set(2), … Set(N_S). Let N_{WS} be the number of locations of each set of the associative-cache (i.e. the number of ways); N_{WB} be the number of words of each virtual memory block; N_d the number of data bits of each memory word (which is also the number of bits of the data field of the set-associative cache and of the Overflow CAM). Let N_{t1} be the number of bits of the tag field of the set-associative cache; N_{t2} be the number of bits of the tag field of the Overflow CAM; and N_f be the number of flag bits in each location of the set-associative cache and of the Overflow CAM.
As we have seen earlier, in ECC-based repair, the probability for a word of the memory to be good (i.e. to contain 0 or 1 faulty cells) is:

$$P_{WMG} = (\mathbf{1} - P_f)^{Nd} + Nd(\mathbf{1} - P_f)^{Nd-\mathbf{1}} P_f$$

Also, if the area of the tag cell is q times larger than the area of the SRAM cell and the area of the flag cell is r times larger than the area of the SRAM cell, then, the probability for a location of the set-associative cache to be good is:

$$P_{WSG} = (1 - P_f)^{(qNtl + rNf)}((1 - P_f)^{Nd} + Nd(1 - P_f)^{Nd-1} P_f).$$

The probability for a location of the Overflow CAM to be good is:

$$P_{WOG} = (1 - P_f)^{(qNt2 + rNf)}((1 - P_f)^{Nd} + Nd(1 - P_f)^{Nd-1} P_f).$$

The probability that a set of the set-associative cache repairs all the faulty words in the corresponding block of the memory (i.e. 0 words are left unrepaired) is:

$$P_{\mathbf{0}UF} = \sum_{t=\mathbf{0}}^{N_{WS}} \left(\frac{N_{WB}! P_{WMG}^{(N_{WB}-t)}}{(N_{WB}-t)!t!} (\mathbf{1} - P_{WMG})^t \sum_{r=\mathbf{0}}^{N_{WS}-t} \frac{N_{WS}! P_{WSG}^{N_{WS}-r}}{(N_{WS}-r)!r!} (\mathbf{1} - P_{WSG})^r \right) \quad \mathbf{(2)}$$

The probability that a set of the set-associative cache leaves unrepaired exactly k faulty words in the corresponding block of the memory is:

$$P_{kUF} = \sum_{t=k}^{N_{WS}+k} \left(\frac{N_{WB}! P_{WMG}^{(N_{WB}-t)}}{(N_{WB}-t)!t!} (\mathbf{1} - P_{WMG})^t \frac{N_{WS}! P_{WSG}^{(t-k)}}{(N_{WS}-t+k)!(t-k)!} (\mathbf{1} - P_{WSG})^{(N_{WS}-t+k)} \right) \quad \mathbf{(3)}$$

Let k(1), k(2), … k(N_S) be the number of words of the N_S virtual memory blocks MB(1), MB(2), … MB(N_S) that are left unrepaired respectively by sets Set(1), Set(2), … Set(N_S) of the set-associative Cache.
The probability that k(1) words in MB1, k(2) words in MB2, … and k(N_S) words in MBN_S are unrepaired is equal to: $P_{k(1)UF} P_{k(2)UF} \ldots P_{k(Ns)UF}$, where the values of the probabilities $P_{k(i)UF}$, are computed by the expression (2) for k(i) = 0 and by expression (3) for k(i) > 0.
In order to repair these faults, the Overflow CAM must dispose at least Q = k(1) + k(2) + … k(N_S) good words.
Let N_{WO} be the number of words of the Overflow CAM. The probability that the Overflow CAM has at least Q fault-free words is:

$$P_{QCO} = \sum_{u=\mathbf{0}}^{N_{WO}-Q} \frac{N_{WO}! P_{WOG}^{(Nwo-u)}}{(N_{WO}-u)!u!} (\mathbf{1} - P_{WOG})^u \qquad \mathbf{(4)}$$

where $P_{WOG} = (1 - P_f)^{(2.8Nt2 + 2Nf)}((1 - P_f)^{Nd} + Nd(1 - P_f)^{Nd-1} P_f)$ is the probability for a location of the Overflow CAM to be good, as determined earlier.
Then, the probability that the memory is repaired when k(1) words in MB1, k(2) words in MB2, … k(N_S) words in MBN_S are left unrepaired by the set associative cache:

$PR_{k(1),k(2),\cdots k(Ns)} = P_{QCO}P_{k(1)UF}P_{k(2)UF}\ldots P_{k(Ns)UF}$ **(5)**

where $k(1) + k(2) + \ldots k(N_S) = Q$, $0 \leq k(i)\ \forall\ i \in \{0, 1, \ldots N_S\}$, and P_{QCO} is computed from (4).

For $Q = k(1) + k(2) + \ldots k(N_S) > N_{WO}$ we have $P_{QCO} = 0$. Thus, we only have to consider the cases where $Q = k(1) + k(2) + \ldots k(N_S) \leq N_{WO}$. Therefore, to compute the total probability for the memory to be repaired, we have to sum the probabilities $PR_{k(1),k(2),\cdots k(Ns)}$ for all possible value combinations of N_S positive integers k(1), k(2), … k(N_S) having sum equal to Q, for all $Q \leq N_{WO}$.

In number theory, the combinations of Ns positive integers having sum equal to Q are referred as the compositions of Q into N_S parts. The number of compositions of length N_S of integer Q is known to be equal to $C'_{Ns(Q)} = (Q+ N_S-1)!/Q!(N_S -1)!$ which will give huge numbers in many practical cases. For instance, for $N_S = 64$ and $Q = 32$, which will be required when we use a set associative cache having 64 sets and 32 ways, we find $C'_{Ns(Q)} \approx 1{,}9801165182011 \times 10^{25}$, which is a huge number of compositions and computing the corresponding probabilities cannot be done at reasonable computation time. So, we need a more efficient approach. The solution to this complex problem, together with fast yield computation algorithms for the Overflow CAM repair architecture is presented in chapter 5.

4.3 EVALUATIONS

In this section we evaluate the efficiency of the proposed architectures. First we use the analytical computation approach shortly described in the previous section and developed in chapter 5, to determine the sizes of the caches/CAM enabling significant reduction of the number of parallel comparisons. Then, we use CACTI [34][56][57] to evaluate the area and power of each solution. The results are shown in tables 1 and 2.

In both tables, column 1 gives the defect densities (expressed as the probability of a memory cell to be faulty). SOCs comprising a total of 9,75 Gbit SRAM capacity (i.e. a total of 250M words x 39 bits per word corresponding to 32 data bits and 7 Hamming code bits) are considered. This memory capacity is distributed over embedded memories 300 and 3000, as reported in column 2. In the results presented in tables 1 and 2, the target yield for the total memory capacity is 90%.

Table 1 concerns conventional (i.e. non-ECC) repair.

Columns 3, 4, and 5 provide results for conventional repair using a CAM: column 3 presents the number of CAM words required to reach the target yield (i.e. 90%), columns 4 and 5 give the area and the power penalties.

Columns 6 to 11 provide results for the new repair architecture using the Overflow Set-Associative Cache repair approach, which employs two caches (CACHE 1 and CACHE 2): columns 6 and 7 give the number of sets and the number of ways of CACHE 1; columns 8 and 9 give the similar numbers for CACHE 2; columns 10 and 11 give the area and power penalties.

We observe in table 1 that for conventional repair, the new repair architecture achieves drastic power reduction at the expense of slight increase of the area penalty.

Table 1. Size, area, and power for non-ECC repair.

P_f	#Emb Mem	Non-ECC Repair CAM			Non-ECC Repair CACHE-1 / CACHE-2					
		N_{CW}	%A	%P	N_{S1}	N_{W1}	N_{S2}	N_{W2}	%A	%P
10^{-4}	300	3466	1.32	185.3	64	63	2	39	1.879	22.73
	3000	402	1.27	67.90	32	18	**1**	1**3**	2.376	23.94

3x 10^{-4}	300	10285	3.93	532.9	128	99	2	30	4.548	33.23
	3000	1121	3.46	177.9	64	24	2	20	6.251	42.33
10^{-3}	300	35325	12.75	1629	512	85	64	32	15.20	65.16
	3000	3693	13.49	581.5	128	39	2	27	15.19	70.33

Table 2 gives the similar results for ECC-based repair. In this table no results are reported for the new architecture in the cases where a small CAM is required for the CAM-based repair architecture (improvements are marginal in these cases).

Note that, the size of the run-time CAM used in all ECC-based repair approaches developped in chapters 2 and 3 is the same (in all cases the size of this CAM is determined in order to repair the memory words containing two or more faulty cells). Thus, run-time power dissipation is the same for all of them. On the other hand, the total area penalty is not the same, since the approaches proposed in chapter 3 use a diagnosis-CAM and a runtime-CAM, while the approach proposed in chapter 2 uses only runtime CAM. As the present paper targets the reduction of runtime power, tables 1 and 2 report the area and power penalties for the run-time CAM/Caches, which is the same for all ECC-based repair schemes proposed in chapters 2 and 3.

We observe that, the new repair architecture (CACHE 1 / CACHE 2) achieves significant power reduction in most cases, leading to low power penalty for high defect densities ($P_f = 10^{-4}$ and $P_f = 3x10^{-4}$), and moderate power penalty (less than 10%) for very high defect densities ($P_f = 10^{-3}$). In addition, a promising technique presented in section 4.4, should enable further power reduction.

Table 2. Size, area, and power for ECC-based repair.

P_f	#Emb Mem	ECC Repair CAM			ECC Repair CACHE-1 / CACHE-2					
		N_{CW}	%A	%P	N_{S1}	N_{W1}	N_{S2}	N_{W2}	%A	%P
10^{-4}	300	16	0.008	1.267	4	8	-	-	1.201	0.017
	3000	6	0.028	1.297	-	-	-	-	-	-
3x 10^{-4}	300	83	0.036	5.337	16	6	1	12	0.069	3.723
	3000	17	0.078	3.676	-	-	-	-	-	-
10^{-3}	300	720	0.249	39.56	64	14	2	30	0.544	9.68
	3000	98	0.344	17.56	16	8	1	10	0.646	9.93

Note finally that CACTI does not allow implementing each set of a K-way set-associative cache by means of a CAM of K locations (CAM-tag architecture). Instead it uses K 1-way set associative cache blocks (RAM-tag architecture). Search in such cache architectures requires selecting one location in each of these K blocks, reading concurrently all of them (i.e. K locations) and comparing their tag fields with the corresponding bits of the current address value. For highly associative CAMs (i.e. using large values of K), this architecture is extremely power hungry [38]. Thus, in our evaluations, we used artifacts to divert CACTI in order to estimate the area and power of the CAM-tag architectures for the set associative caches used in our experiments. We are looking to estimate the area and power of a set-associative cache comprising V sets of K-ways each, and implemented by employing CAM-tag architecture. This estimation was done by using the following steps:

1. Use CACTI to evaluate the area and power of a CAM of K locations (implementing one set of the set associative cache).

2. Use CACTI to determine the number locations of a 1-way cache having the same number of data and tag bits as the CAM in step 1, occupying the same area as this CAM, and having as closely as possible the same form factor.
3. Use CACTI to implement a 1-way cache partitioned into V banks, with each bank being identical to the 1-way cache of step 2.
4. The searched area for the CAM-tag set-associative cache comprising V sets of K-ways is taken equal to the area of the cache of step 3. This is because each bank used in the cache of step 3 was taken to have the same area as each set of CAM-tag set-associative cache. In addition: the two caches have V banks of equal size; use the same number of address bits for selecting one of these blocks; and have to route the same number of tag and data bits to each of these blocks. Thus, the area for address-decoding and routing will also be the same.
5. The searched power for the CAM-tag set-associative cache is obtained by subtracting from the power of the cache of step 3 the power of the circuit of step 2 and adding the power of the CAM of step 1. This is because, for similar reasons as in step 4, the power for address-decoding and routing of the CAM-tag set associative cache is the same as for the cache of step 3, and the power of each set of the CAM-tag set-associative cache is equal to the power of the CAM of step 1.

Also, while the above approach is best suited for estimating the area and power of a CAM-tag set-associative cache by means of CACTI, the obtained power estimations are still pessimistic due to various implementation options used in CACTI [38]. Thus, using dedicated implementations for CAM-tag set-associative cache architectures should allow in the future significant power reduction with respect to the results presented in tables 1 and 2.

Note finally that, due to the artifacts used for estimating the area and power of CAM-tag set-associative caches by means of CACTI, and also because the number of sets is always a power of 2 (resulting in discontinuous choices in the number of sets), we cannot obtain smooth reduction of power penalty. Thus, we may observe certain apparent anomalies concerning the evolution of the area and power penalties for certain RAM sizes and defect densities. For instance, in table 2, for the case of $P_f = 10^{-4}$ and 300 embedded memories, the new repair architecture reduces the power penalty from 1.267% to 0.017%; while, for the case of $P_f = 10^{-4}$ and 3000 embedded memories, no power reduction is obtained.

4.4 Overflow CAM/Cache Conditional Selection

As shown in the previous sections, the proposed partitioning-based repair architectures enable dramatic reduction of power dissipation. Low power dissipation is however a stringent requirement in modern technologies. Thus, additional power reduction is always welcome. In this section we present a promising concept for additional power reduction, whose evaluation and validation was not yet realized due to time constraints, but will represent in our future developments, one of the promising extensions of the presented work.

In the Overflow CAM repair architecture of figure 2, in any memory operation d address bits select one set of the set-associative cache over which the t bits of the target address are searched (where d+t = N@ are the address bits of the memory). In addition the d+t bits of the target address are searched over the Overflow CAM. Thus, the dynamic power dissipation of the repair circuitry has two main sources, the selected set of the set-associative cache and the Overflow CAM. The Overflow CAM may contain any memory address. Therefore, we are obliged to perform searches in the Overflow CAM during every memory operation. However, with a careful distribution of the faulty memory addresses in the sets of the set-associative cache and the Overflow CAM, we can restrict the space of addresses allocated to the Overflow CAM and use the CAM selectively to reduce its power dissipation.

This can be done by means of the "Rearrangement Process" described bellow, which swaps memory addresses stored in the set-associative cache with memory addresses stored in the Overflow CAM, in a manner that concentrates in the Overflow CAM the higher order addresses. By checking few address bits this rearrangement allows disabling most of the time the Overflow CAM, reducing significantly its power dissipation. A way for making this rearrangement is to use a sorting algorithm for organizing in ascending order the addresses stored in sets of the set-associative cache and in the overflow CAM. However, sorting algorithms are relatively complex in terms of: hardware (if they are implemented by dedicated hardware rather than an existing processor core); execution time; and power dissipation. Thus, in the following we propose a much simpler process (referred as "Rearrangement Process"), which creates a loose ordering of the addresses in the set associative Cache and the Overflow CAM without affecting power reduction efficiency, and requires simple hardware, short execution time, and low power dissipation.

"Rearrangement Process" employs a d-bits counter Cnt1 (where d is the number of address bits used for selecting the sets of the set-associative cache); two binary counters Cnt3 and Cnt4 of size equal to $\log_2\lceil N_{WO} \rceil$; a k-bits shift-register SR_k able to perform left shifts with a 0 entering its rightmost bit at each shift, where the value of k depends on the memory size, the parameters of the partitioning-based repair architecture, and the defect density, and in most practical situations will not exceed 7[13]; a register REG1 of size equal to the size of the tag field of the overflow CAM, and a simple controller, which controls these circuits, the *FLC* counters of the set-associative cache and the Overflow CAM (see section 3.1 about the function of the *FLC* counter), and the operations of the set-associative cache and the Overflow CAM, in order to execute the "Rearrangement Process".

Below we describe the "Rearrangement Process", where we consider that the d address bits used for set selection (the set-selection bits) are the less significant address bits of the memory.

The rearrangement process tries to move in the Overflow CAM bad words having k 1's in their k most significant address bits, and if this does not succeed it tries for k-1 1's in the k-1 most significant address bits by entering a 0 on the rightmost bit of SR_k, and so on.

Rearrangement Process

We set to 1 all bits of SR_k, we reset Cnt1, and we go to the "Counting Part of the Rearrangement Process".

Counting Part of the Rearrangement Process

We reset Cnt3 and Cnt4.

i. We access sequentially the locations of the Overflow CAM by incrementing its *FLC* counter, and we read from each location the tag field and the flag bits.

ii. For each read CAM location in which flag1 is 0 (good CAM location) and flag2 is 0 (CAM location occupied for storing a repaired memory word), we compare the d less significant tag bits against the content of Cnt1, and we check if the k most significant tag bits have 1 in each position SR_k has 1.

iii. Each time the comparison against the contents of Cnt1 matches we increment counter Cnt3.

iv. Each time the comparison against the contents of Cnt1 matches and the check against SR_k succeeds, we increment counter Cnt4.

[13] The total number of bits of these counters and of the shift register will not exceed 24 in most practical applications, representing a very low area cost.

v. We use the d bits of Cnt1[14] as set-address for selecting the corresponding set of the set-associative cache.

vi. We access sequentially the locations of the set-associative cache by incrementing its *FLC* counter, and we read from each location its tag field and flag bits.

vii. For each location in which flag1 is 0 and flag2 is 0, we check if the k most significant tag bits have 1 in each position in which SR_k has 1.

viii. Each time the check against SR_k succeeds, we increment counter Cnt4.

ix. If at the end of this process the content of Cnt4 is lower than the content of Cnt3: we perform a left-shift on SR_k; and we re-execute the "Counting Part of the Rearrangement Process".

x. When the content of Cnt4 is equal to or larger than the content of Cnt3, we go to the "Rearrangement Part of the Rearrangement Process**"**.

Rearrangement Part of the Rearrangement Process

i. We access sequentially the locations of the Overflow CAM by incrementing the *FLC* counter of this CAM, and we read from each location its tag field and flag bits.

ii. For each location in which flag1 is 0 and flag2 is 0 we check if the k most significant tag bits have 1 in each position in which SR_k has 1.

iii. If this check fails, we read the tag field of this location and we store it in register REG1.

iv. We use the d bits of Cnt1[15] as address for selecting the corresponding set of the set-associative cache.

v. We use the *FLC* counter of the set-associative cache to access sequentially the locations of this set, and we read from each location its tag field and flag bits.

vi. For each location in which flag1 is 0 and flag2 is 0, we check if the k most significant tag bits have 1 in each position in which SR_k has 1.

vii. In the first location in which this check succeeds we read the tag field; we concatenate it with the content of Cnt1 (placing this content on the right side); and we store this concatenation in the tag field of the location of the Overflow CAM selected by the current value of the *FLC* counter of this CAM;

viii. We store the (N@ – d) leftmost bits of REG1 in the tag field of the location of the set-associative cache selected by the current value of the *FLC* counter of this cache, where N@ is the number of address bits of the memory under repair.

ix. If the *FLC* counter of the overflow CAM is full, we increment Cnt1 and we go to the "Counting Part of the Rearrangement Process", to treat the next set of the set-associative cache.

x. The "Rearrangement Process" ends when we finish the treatment of the last set of the set-associative cache (i.e. the one selected by the highest value of Cnt1, i.e. all bits of Cnt1 are equal to 1).

Let us now discuss the results of the above process.

Outcome of the "Rearrangement Process"

[14] plus the d2 bits of another counter Cnt2 in the case of the Overflow Set-Associative Cache repair architecture as described later.

[15] Same comment as in footnote 2.

Let us consider d bits and any value V_d of these bits, and the execution of "Rearrangement Process" for Cnt1 = V_d. Then we find:

a. From the step iii of the "Counting Part of the Rearrangement Process" we find that at the end of this process the state of Cnt3 is equal to the number of locations of the Overflow CAM storing memory words having their d less significant bits equal to V_d.

b. From the steps iv and viii of the "Counting part of Rearrangement Process" we find that Cnt4 counts the number of locations of the Overflow CAM and of the set-associative cache that store memory addresses having their d less significant bits equal to V_d, and have in their k most significant bits a 1 in each position in which the current state of SR_k has 1.

c. From the steps ix and x of the "Counting part of Rearrangement Process", we have that Cnt4 is equal to or larger than the content of Cnt3.

d. From points a., b., and c., "The number of locations of the Overflow CAM and of the set-associative cache that store memory words, which have their d less significant bits equal to V_d, and have in the k most significant bits of their tag field a 1 in each position in which the current state of SR_k has 1" *is equal to or larger than* "the number of locations of the Overflow CAM storing memory words having their d less significant bits equal to V_d".

e. From the above relation, the execution of the "Rearrangement Part of the Rearrangement Process" for each value to V_d of Cnt1, guaranties that, at the end of this execution, all memory addresses that are stored in the Overflow CAM and have their d less significant bits equal to V_d, have also in their k most significant bits a 1 in each position in which the current state of SR_k has 1".

f. Statement e. is valid for all possible values V_d of d bits.

g. The initial state of SR_k is 111 … 11, and it is modified by means of left-shifts (step ix of the "Counting Part of the Rearrangement Process"), which enter a 0 in the rightmost bit of SR_k. Thus, any earlier state of SR_k has 1 in each position in which the final state of SR_k has 1.

h. From points e., f., and g., we find that: all memory addresses stored in the Overflow CAM have in their k most significant bits a 1 in each position in which the final state of SR_k has 1.

From point h., we can use the final state of SR_k to select the Overflow CAM by checking if the k most significant bits of the address of the current memory operation have 1 in each position SR_k has 1. Then, if this check succeeds we select the Overflow CAM, and if it fails we do not select this CAM. Thus, if the final state of SR_k contains r 1s, the Overflow CAM will be selected once at every 2^r memory operations. Thus, at the mean, its power dissipation will be divided by 2^r.

This approach can also be used in the case of the Overflow Set-Associative Cache repair architecture. Indeed, let N be the number of words of the memory under repair, V1 and K1 be respectively the numbers of sets and of ways of CACHE 1, and V2 and K2 be respectively the number of sets and of ways of the Overflow Set-Associative Cache (CACHE 2). Then, we can consider that we have V2 virtual memories such that: each of these virtual memories comprises N/V2 memory words, and is repaired by means of a virtual set-associative cache having V1/V2 sets of K1 ways each, and a virtual Overflow CAM of K2 locations (i.e. corresponding to a set of CACHE2). Thus, we can employ the approach described above, to each of these V2 virtual memories repaired by a virtual set-associative cache and a virtual Overflow CAM, with few modifications as described below.

As we have two set-associative caches (CACHE 1 and CACHE 2), we use two groups of set-selection bits. The one consists in the d1 leftmost bits of the memory address and is used to select the sets of CACHE1, and the other consists in the d2 leftmost bits of the memory address and is used to select the sets of CACHE2 (with d2 < d1 since CACHE1 has more sets than CACHE2). In correspondence with these bits, the "Rearrangement Process" will use a d2-bits counter Cnt2 and a d-bits counter Cnt1 (with d = d1-d2). The content of Cnt2 will be used for selecting the sets of CACHE2, while the concatenation of Cnt1 and Cnt 2 (with Cnt1 on the right of Cnt2), will be used for selecting the sets of CACHE1.

To perform the "Rearrangement" in the Overflow Set-Associative Cache repair architecture, Cnt1 will be incremented to select successfully each set of CACHE2, and for each of these sets the "Rearrangement Process" will be executed as described earlier, with only difference the use of the concatenation of Cnt1 and Cnt2 for selecting the sets of CACHE1, as explained above and reported in the footnotes inserted in the description of the "Rearrangement Process".

At the end of the "Rearrangement Process", for each of the V2 virtual memories and the virtual set-associative cache and virtual Overflow CAM repairing it, we will dispose a SR_k shifter for each virtual Overflow CAM (i.e. for each set of CACHE2). Then, the content of each of these shifters will be used to select the corresponding set of CACHE2. That is, each set of CACHE2 will be selected by decoding the d2 less significant address bits of the memory and by checking the compliance of the k most significant address bits to the content of the corresponding SR_k shifter, dividing by a significant factor the power dissipation of CACHE 2. Note also that, as we dispose an individual SR_k shifter for each set of CACHE2, we do not have to use for each set of CACHE2 the disabling condition of the worst case set. Instead, we use for each set of CACHE2 its optimal disabling condition, as determined by the "Rearrangement Process" in response to: the number of faulty locations of this set of CACHE2 (virtual Overflow CAM); the number of faulty locations of the corresponding virtual set-associative cache; and the distribution of the faulty memory words in the address space of the corresponding virtual memory.

4.5 CONCLUSION

In the previous chapters we addressed the issue of the diagnosis CAM required in ECC-based repair. For high defect densities, the proposed solutions enable reducing drastically area and power penalties with respect to conventional repair. However, power penalty is still significant under the stringent requirements for low power in advanced technologies. Thus, to achieve further power reduction, in this chapter we propose new repair architectures using set-associative caches at multiple levels, which allow significant additional power reduction. At the same time, yield estimation is becoming complex due to the introduction of multiple repair-levels. Thus, in this chapter are also developed new yield estimation mathematics and related algorithms to handle these cases. The evaluation of the new architectures by using these algorithms, as well as the CACTI framework, shows that they can be advantageously combined with the developments of chapters 1, 2, 3, and 4, to achieve both low area and low power penalties even in the case of very high defect densities.

CHAPTER 5

Yield Computation Mathematics for Memory Repair Architectures

To evaluate the memory repair approaches proposed in chapters 1, 2, 3, and 4, we need to determine the size of the CAMs/Caches required for achieving a target yield for ECC-based repair and non-ECC repair implemented by means of the proposed architectures. The analytical computation of the yield becomes increasingly complex for multiple reasons:

- In low defect densities, the faults affecting the repair CAM have negligible impact on the yield. But in high defect densities this impact is significant, requiring more complex yield computation.
- In high defect densities the size of the repair CAM becomes very high, increasing drastically the number of operations required to compute the yield.
- The introduction of sophisticated memory repair architectures, as the ones presented in chapter 4, requires very complex yield computation expressions.

Due to the above issues, we need to develop yield computation mathematics, enabling computing the yield in reasonable time.

Note also that, the fault injection approach is an alternative to the analytical yield computation. However, the computation time of this approach too becomes very high due to the following reasons:

- As in advanced technologies we have to consider very large memories, the duration of each fault injection experiment also increases.
- In sophisticated repair architectures, as the set-associative cache architectures proposed in chapter 4 for reducing power dissipation, we need to reduce the size of each cache set, and thus the size of the corresponding virtual memory. Decreasing the number of words of each virtual memory, decreases the population of the words repaired by each cache set. As population reduction increases the standard deviation of a statistical distribution, we will observe an increase of the deviation of the number of faulty memory words affecting the virtual memory blocks. Thus, to obtain statistical significance we need to perform much larger numbers of fault injections in comparison with the classical repair architectures.

In addition, the analytical computation approach is of higher precision as it guaranties exact yield computation. Thus, in this chapter we investigate the mathematics of yield computation, in order to achieve exact yield computation in very short time.

5.1 FAST YIELD COMPUTATION FOR CAM-BASED REPAIR, AND SET-ASSOCIATIVE CACHE REPAIR

Among the repair architectures developed in the previous chapters, the CAM-based repair architecture requires the simpler analytical expressions for yield computation. In addition these expressions can be used for computing the yield for the set-associative cache repair architecture. Indeed, in this case we can consider that we have a system of M memories, where M is the number of virtual memory blocks repaired

by the corresponding sets of the set-associative cache. Thus, we can compute the yield Y for each virtual memory block by considering it as a memory repaired by a CAM having a number of locations equal to the number K of locations of each set of the set-associative cache. Then, the global yield of the memory will be given by $Y_{MEM} = Y^M$.

We can compute the yield for conventional repair and ECC-based repair in the following manner. Let P_f be the probability of a memory cell to be faulty, and P_{wg} be the probability that a memory word does not need to be repaired (good word). In conventional repair, Pwg is equal to the probability that the memory word is fault-free and can be computed as $P_{wg} = (1 - P_f)^N$, where N is the number of bits of the memory word. In ECC-based repair P_{wg} will give the probability that the memory word is fault-free or it contains one faulty cell. Thus, P_{wg} can be computed as $P_{wg} = (1 - P_f)^N + N(1 - P_f)^{N-1}P_f$.

A CAM location comprises a tag field, a data field (corresponding to a memory word), and few flag cells. The data field, the tag field, and the flag cells are implemented with SRAM cells, but the tag field also integrates a comparator. Let the tag-cell area be r times larger than the SRAM cell and the flag cell area be q times larger than the SRAM cell. Then, the probability that the tag field does not contain faulty cells is equal to $(1 - P_f)^{rN@}$, where $N_@$ is the number of cells of the tag field, and the probability that the flag cells are fault-free is equal to $(1 - P_f)^{qNf}$, where N_f is the number of flag cells. Most authors consider the value of r and q to be about 2, and this is also the case in the CACTI code. Thus we consider this value in our computations.

Let Pwcg be the probability that a CAM location is good for repairing a faulty memory word. In conventional repair, Pwcg is the probability that the CAM location is fault-free. Thus we have $P_{wcg} = (1 - P_f)^{rN@ +qNf+N}$. In ECC-based repair, Pwcg is the probability that the tag field and the flag bits are fault free, and the data field is fault-free or it contains one faulty cell. Thus we have $P_{wcg} = (1 - P_f)^{rN@+qNf}((1 - P_f)^N + N(1 - P_f)^{N-1}P_f)$.

Let N_W be the number of words of the memory and N_{WC} be the number of words of the repair CAM. Then, considering uniform fault distribution we find easily that the sum $\sum_{r=0}^{N_{wc}-t} \frac{N_{wc}!P_{wcg}^{N_{wc}-r}}{(N_{wc}-r)!r!}(1-P_{wcg})^r$ gives the probability that there are at least t good CAM locations. Then, the expression $\frac{N_w!P_{wg}^{(N_w-t)}}{(N_w-t)!t!}(1-P_{wg})^t \sum_{r=0}^{N_{wc}-t} \frac{N_{wc}!P_{wcg}^{N_{wc}-r}}{(N_{wc}-r)!r!}(1-P_{wcg})^r$ gives the probability that there are t memory words requiring repair and at least t good CAM locations (i.e. the probability that the memory contains t faulty words and is repaired). Then, as the number of faulty memory words that can be repaired cannot exceed the number N_{WC} of the CAM words, the memory yield after repair is obtained by summing this expression from t=0 to t=Nwc, resulting in the expression:

$$Y = \sum_{t=0}^{N_{wc}} \left(\frac{N_w!P_{wg}^{(N_w-t)}}{(N_w-t)!\,t!}(1-P_{wg})^t \sum_{r=0}^{N_{wc}-t} \frac{N_{wc}!P_{wcg}^{N_{wc}-r}}{(N_{wc}-r)!\,r!}(1-P_{wcg})^r \right) \quad (1)$$

Expression 1 is valid for both conventional repair and ECC-based repair, provided that the probabilities of good memory word (Pwc) and good CAM word (Pwcg) are computed as discussed earlier for the conventional repair and for the ECC-based repair.

To evaluate the computation complexity of expression (1) we need to determine the number of operations required for its computation. To determine this number we consider that, the factorials in the nominator and the denominator in the two fractions of expression (1) are simplified by eliminating the one factorial in the denominator as well as all the terms of this factorial from the factorial in the nominator. The optimal manner to simplify these factorials is to eliminate from the denominator the factorial (Nw –t)!, if (Nw –t) ≥ t, or the factorial t! otherwise, and do the similar for (Nwc –r)! and t!. Also, let k be the number of terms

(without considering the term 1) of the factorial left in the denominator. In the computations we can either perform k-1 multiplications to compute the product of these terms and then divide the value of the nominator by the value of this product (requiring k-1 multiplications and 1 division); or divide successively the value of the nominator by each of these terms (requiring k divisions). Next we use the former manner as it minimizes the number of divisions.

Derivation of the number of multiplications:

The number of multiplications required for the term $C_t = \frac{N_w! P_{wg}^{(N_w - t)}}{(N_w - t)! t!}(1 - P_{wg})^t$ is (t-1) +(Nw-1) + (t-2) + 1 = (Nw-3) +2t for all values of t ***except for t=0 for which we should add 2, and for t = 1 for which we should add 1***. Thus, summing them for t= 0 to t = Nwc gives (Nwc +1)(Nw – 3) + 2(0 + 1 + 2 + … Nwc) + 2 + 1 (where +2 and + 1 are the corrections for t = 0 and t = 1. Thus we obtain (Nw – 3)(Nwc +1) + Nwc(Nwc +1) + 3 multiplications.

The number of multiplications required for the term $B_r = \frac{N_{wc}! P_{wcg}^{N_{wc} - r}}{(N_{wc} - r)! r!}(1 - P_{wcg})^r$ is (r-1) +(Nwc-1) + (r-2) + 1 = (Nwc-3) +2r for all values of r **except for r=0 for which we should add 2 and for r = 1 for which we should add 1, as well for the case r = Nwc (which can occur only once – when t = 0) for which we should subtract 1 from the final result. Also, when t = Nwc, r takes only the value 0. Thus, we have to subtract 1 from the final result, to compensate the + 1 added for the case r=1, which does not occur when t = Nwc.**

Thus, summing (Nwc-3) +2r for r= 0 to r = Nwc-t gives (Nwc –t +1)(Nwc – 3) + 2(0 + 1 + 2 + … + Nwc - t) + 2 + 1 (where +2 and + 1 are the corrections for t = 0 and t = 1. Thus we obtain (Nwc -t +1)(Nwc – 3) + (Nwc -t)(Nwc -t +1) + 3 = (Nwc +1)(2Nwc – 3) - t(3Nwc – 2) +t^2 + 3 multiplications, **where the +3 is for the corrections of r=0 and r = 1**. Then, summing for t = 0 to t = Nwc and **subtracting 2 to compensate for the cases r = Nwc, and t = Nwc for which r=1 does not occur**, we obtain (Nwc +1)^2(2Nwc – 3) - (3Nwc – 2)Nwc(Nwc +1)/2 + Nwc(Nwc +1)(2Nwc +1)/6 + 3(Nwc +1) – 2.

Summing the multiplications required for the terms Ct and Br, plus Nwc + 1 times the multiplication of Ct by $\sum_{r=0}^{N_{wc} - t} B_r$ gives: (Nw – 3)(Nwc +1) + Nwc(Nwc +1) + 3 + (Nwc +1)^2(2Nwc – 3) - (3Nwc – 2)Nwc(Nwc +1)/2 + Nwc(Nwc +1)(2Nwc +1)/6 + 3(Nwc +1) – 2 + (Nwc + 1) = Nw(Nwc +1) + (Nwc^2 – 1)(5Nwc +12)/6 +1 multiplications.

Derivation of the number of divisions:

Each term Ct requires one division, giving Nwc + 1 divisions for all terms Ct (i.e. for t = 0 to t = Nwc).

Each term Br requires 1 division. Thus, the sum $\sum_{r=0}^{N_{wc} - t} B_r$ requires Nwc + 1 – t divisions. Then, summing Nwc + 1 – t for t = 0 to t = Nwc, to take into account the external sum, gives (Nwc + 1)(Nwc +1) – Nwc(Nwc +1)/2. Adding to this number Nwc + 1 (the divisions for all terms Ct), we find a total number of divisions equal to (Nwc + 1)(Nwc + 4)/2.

Derivation of the number of Additions:

Nwc – t additions are required for summing the Nwc – t + 1 terms Br of the internal sum $\sum_{r=0}^{N_{WC}-t} B_r$ of expression 1. From the external sum $\sum_{t=0}^{N_{WC}}$ of expression 1 there are Nwc + 1 internal sums (from t = 0 to t = Nwc). Thus, summing Nwc – t + 1 from t = 0 to t = Nwc gives Nwc(Nwc +1) - Nwc(Nwc +1)/2 additions. Adding the Nwc additions for summing the Nwc + 1 terms of the global sum $\sum_{t=0}^{N_{WC}}$ gives a total of $N_{WC}(N_{WC}+3)/2$ additions.

The outcome is that the computation of expression (1) requires: **$Nw(Nwc + 1) + (Nwc^2 - 1)(5Nwc + 12)/6 - 1$ multiplications; $(Nwc + 1)(Nwc + 4)/2$ divisions; and $N_{WC}(N_{WC} + 3)/2$ additions**.

This complexity is much higher with respect to the yield computation for low defect densities, where faults affecting the CAM have insignificant impact to the yield and are ignored, resulting in much simpler yield computation expression: $Y = \sum_{t=0}^{N_{WC}} \frac{N_w! P_{wg}^{(N_W - t)}}{(N_w - t)!\, t!} (1 - P_{wg})^t$

Also, as we deal with future very advanced technologies allowing producing very complex chips, we should be able to deal with very large memories. Moreover, as we deal with high defect densities we should be able to deal with large repair CAMs. In this context, the above numbers of operations are too large. For instance, for conventional repair of a 10 Gbit memory employing 32-bits words and affected by a 10^{-3} fault density, we have Nw = 335544320, and Nwc ≈ 350000. Thus, computing the yield by means of (1) requires 3.58×10^{16} multiplications, plus 6.12×10^{10} divisions and the similar number of additions. Furthermore, these operations have to manipulate very large numbers such as Nw!/(Nw-t)!t!, as well as very small numbers such as P_{wcg}^{Nw}, requiring high precision arithmetic. Thus, computing the yield by means of expression (1) becomes computationally intractable.

To accelerate the computation of expression (1) we discovered certain recursive relations described bellow, which, to the best of our knowledge are unknown in the literature.

Setting $A_t = \frac{N_W! P_{Wg}^{(N_W - t)}}{(N_W - t)!\, t!} (1 - P_{Wg})^t$, $B_r = \frac{N_{WC}! P_{WCg}^{N_{WC} - r}}{(N_{WC} - r)!\, r!} (1 - P_{WCg})^r$, we can write expression (1) as:

$$Y = \sum_{t=0}^{N_{WC}} \left(A_t \sum_{r=0}^{N_{WC}-t} B_r \right) \qquad (2)$$

We find that At and Br can be written recursively as:

$$A_0 = P_{Wg}^{N_W}, \quad A_{t+1} = \frac{(N_W - t)(1 - P_{Wg})}{(t+1) P_{Wg}} A_t \qquad (3)$$

$$B_0 = P_{WCg}^{N_{WC}}, \quad B_{r+1} = \frac{(N_{WC} - r)(1 - P_{WCg})}{(r+1) P_{WCg}} B_r \qquad (4)$$

We also set $B'_t = \sum_{r=0}^{N_{WC}-t} B_r$ (**5**)

and we find that B'_t and can be written recursively as: $B'_0 = \sum_{r=0}^{N_{WC}} B_r$, $B'_{t+1} = B'_t - B_{N_{WC}-t}$ (**6**)

Based on these relations the computation is done in the following manner:

- The terms B_0, B_1, B_2, …, B_{Nwc} are used intensively in the computations. Thus, to avoid recomputing them multiple times we compute them once for ever by means of relations (4), and we store them in a lookup table. The creation of this table requires *$4N_{WC}$ -1 multiplications and N_{WC} divisions*.
- For each value of t we use the relations (6) and the stored values of B_0, B_1, B_2, …, B_{Nwc} to compute each term B'_t: The term B'_0 requires N_{WC} additions and each other term B'_t one subtraction, resulting in a total of *N_{WC} additions and N_{WC} subtractions* for computing all the terms B'_t.
- For each value of t we use the relations (3) to compute each term A_t. A_0 requires N_W -1 multiplications and each other term A_t three multiplications and one division, resulting in a total of *N_W -1 + $3N_{WC}$ multiplications and N_{WC} divisions* for computing all the terms A_t.
- We use relations (2), (5) and the values of A_t and B'_t to compute the yield. This will require *N_{WC} + 1 multiplications and N_{WC} additions.*

Note that the values of A_t and B'_t do not need to be computed in advance and stored in look-up tables as they are used only once and we employing them in the good order, i.e. from t = 0 to t = N_{WC} , as required in the recursive relations (3) and (6).

Thanks to the derived recursive relations the computation of the yield can be done in linear complexity with respect to the memory size: **N_W + 8N_{WC} - 1 multiplications, 2N_{WC} divisions, 2N_{WC} additions, and N_{WC} subtractions**. This is dramatically shorter with respect to the number of operations required for computing expression (1) in direct manner. For instance, for the above example of the 10 Gbit memory employing 32-bits words and affected by a 10^{-3} fault density, we can compute the yield by means of $3.36x10^8$ multiplications, $7x10^5$ divisions, $7x10^5$ additions, $3.5x10^5$ subtractions, resulting in a reduction of 8 orders of magnitude. This reduction allows computing the yield in short time even for very large memories and high fault densities.

The recursive yield computation approach for CAM-based repair was implemented in C++, and was used in the previous chapters for evaluating the proposed repair architectures.

5.2 Fast Yield Computation for the Separate-CAMs Architecture

In this section we consider the yield computation for the architecture using two separate CAMs, a diagnosis-CAM used during the test and diagnosis phase and a CAM used at run-time (the runtime-CAM). In this scheme the memory is repaired only if the diagnosis-CAM is able to diagnose the memory and the runtime CAM is able to repair all memory words requiring repair (bad memory words). Thus, to compute the yield in this case we have to *compute the joint probability that the diagnosis-CAM is successful and the CAM is repaired*.

From chapter 3 proposition 2, for the diagnosis process to finish successfully *the diagnosis-CAM must contain at least as many good locations as the number of memory words that need to be repaired.* The probability of this event is given by the expression $\sum_{r=0}^{N_{WDC}-t} \frac{N_{WDC}! P_{WDCg}^{N_{WDC}-r}}{(N_{WDC}-r)!r!}(1-P_{WDCg})^r$ (**7**)

On the other hand, for the repair to be successful, *the runtime-CAM must contain at least as many good locations as the number of memory words that need to be repaired*. As we have seen earlier, the expression

$$\frac{N_W! P_{Wg}^{(N_W - t)}}{(N_W - t)! t!} (1 - P_{Wg})^t \sum_{r=0}^{N_{wc} - t} \frac{N_{WC}! P_{WCg}^{N_{wc} - r}}{(N_{WC} - r)! r!} (1 - P_{WCg})^r \quad \textbf{(8)}$$

gives the probability that the memory contains t words requiring repair and the CAM contains at least *t good* locations (i.e. the probability that the memory contains t words requiring repair and it is repaired). Then, our joint probability is given by multiplying expressions (8) and (7) and taking their sum from t = 0 to t = Nwc, resulting in expression (1). as the number of memory words that can be repaired cannot exceed the number N_{WC} of the CAM words, the memory yield after repair is obtained by summing expression (8) from *t* = 0 to *t* = Nwc, resulting in the following expression:

$$Y = \sum_{t=0}^{N_{wc}} \left(\frac{N_W! P_{Wg}^{(N_W - t)}}{(N_W - t)! t!} (1 - P_{Wg})^t \sum_{r=0}^{N_{wc} - t} \frac{N_{WC}! P_{WCg}^{N_{wc} - r}}{(N_{WC} - r)! r!} (1 - P_{WCg})^r \sum_{r=0}^{N_{WDc} - t} \frac{N_{WDC}! P_{WDCg}^{N_{WDc} - r}}{(N_{WDC} - r)! r!} (1 - P_{WDCg})^r \right) \quad \textbf{(9)}$$

Note that, as the number N_{WDC} of the locations of the diagnosis-CAM is larger than N_{WC}, the memory can be diagnosed successfully even if it contains more than t = N_{WC} bad words. But in expression (9) the external sum is taken until the value t = N_{WC}. Hence, expression (9) does not consider the cases of successful diagnosis in which the memory contains more than N_{WC} bad words. However this is correct as: on the one hand expression (9) gives the joint probability for the memory to be repaired successfully and to be diagnosed successfully, and on the other hand if the memory contains more than N_{WC} bad words, this joint probability is equal to 0 even though the memory can be diagnosed successfully (because the probability that the memory is repaired becomes 0, making 0 the joint probability).

We find that the number of operations required computing the yield by means of (9) is:

$N_{WDC}(N_{WC}+1)(N_{WC}+2)/2 + N_W(N_{WC}+1) + (N_{WC}+1)(7N_{WC}^2 + 2N_{WC} - 6)/6 -$ 3 multiplications; $(N_{WC}+1)(N_{WC}+3)$ divisions; and $N_{WC}(N_{WC}+2)$ additions. Thus the number of operations increases exponentially with the memory and the CAM sizes. In addition, we will have to deal with large values of N_W (as we have to deal with future very advanced technologies allowing producing very complex chips, which will include very large memories), as well as large values of N_{WDC} and N_{WC} (due to the high defect densities), leading to huge numbers of operations. Furthermore, these operations have to manipulate very large numbers such as Nw!/(Nw-t)!t!, as well as very small numbers such as P_{wcg}^{Nw}, requiring high precision arithmetic. Thus, computing the yield by means of expression (1) becomes computationally intractable. To accelerate these computations we discovered certain recursive relations described below, which, to the best of our knowledge are unknown in the literature. Expression (9) can be written as:

$$Y = \sum_{t=0}^{N_{wc}} \left(A_t \sum_{r=0}^{N_{wc} - t} B_r \sum_{r=0}^{N_{WDc} - t} C_r \right) \quad \textbf{(10)}$$

where: $A_t = \frac{N_W! P_{Wg}^{(N_W - t)}}{(N_W - t)! t!} (1 - P_{Wg})^t$, $B_r = \frac{N_{WC}! P_{WCg}^{N_{wc} - r}}{(N_{WC} - r)! r!} (1 - P_{WCg})^r$,

$$C_r = \frac{N_{WDC}! P_{WDCg}^{N_{WDc} - r}}{(N_{WDC} - r)! r!} (1 - P_{WDCg})^r$$

We find that At, Br, and Cr can be written recursively as:

$$A_0 = P_{Wg}^{N_W}, \; A_{t+1} = \frac{(N_W - t)(1 - P_{Wg})}{(t+1) P_{Wg}} A_t \quad \textbf{(11)}$$

$$B_0 = P_{WCg}^{N_{WC}}, \; B_{r+1} = \frac{(N_{WC} - r)(1 - P_{WCg})}{(r+1) P_{WCg}} B_r ,$$

$$C_{\mathbf{0}} = P_{WDCg}^{N_{WDC}},\ C_{r+\mathbf{1}} = \frac{(N_{WDC} - r)(\mathbf{1} - P_{WDCg})}{(r+\mathbf{1})P_{WDCg}} C_r \quad (\mathbf{12})$$

We also set $B'_t = \sum_{r=\mathbf{0}}^{N_{WC}-t} B_r$, and $C'_t = \sum_{r=\mathbf{0}}^{N_{WDC}-t} C_r$ (**13**) and we find that B'_t and C'_t can be written recursively as:

$$B'_{\mathbf{0}} = \sum_{r=\mathbf{0}}^{N_{WC}} B_r,\ B'_{t+1} = B'_t - B_{N_{WC}-t},\quad C'_{\mathbf{0}} = \sum_{r=\mathbf{0}}^{N_{WDC}} C_r,\ C'_{t+1} = C'_t - C_{N_{WDC}-t} \qquad (\mathbf{14})$$

Based on these relations the computation is done in the following manner:

- From relations (12) we compute the terms $B_0, B_1, B_2, \ldots, B_{N_{WC}}$ and $C_0, C_1, C_2, \ldots, C_{N_{WDC}}$ we store them in two lookup tables. This requires $4N_{WC} + 4N_{WDC} - 2$ multiplications and $N_{WC} + N_{WDC}$ divisions.
- For each value of *t* we use the relations (14) and the stored values of $B_0, B_1, B_2, \ldots, B_{N_{WC}}$ and $C_0, C_1, C_2, \ldots, C_{N_{WDC}}$ to compute each of the terms B'_t and C'_t: The term B'_0 requires N_{WC} additions and each other term B'_t one subtraction, resulting in a total of N_{WC} additions and N_{WC} subtractions for computing all the terms B'_t. Similarly for the terms C'_t we need a total of N_{WDC} additions and N_{WDC} subtractions for computing all the terms B'_t.
- For each value of t we use the relations (11) to compute each term A_t. A_0 requires N_W -1 multiplications and each other term A_t three multiplications and one division, resulting in a total of N_W -1 + 3 N_{WC} multiplications and N_{WC} divisions for computing all the terms A_t.
- We use relations (10), (13) and the values of A_t, B'_t, and C'_t to compute the yield. This will require $2N_{WC} + 2$ multiplications and N_{WC} additions.

Thus, the yield computation can be done in linear complexity with respect to the memory size: **$N_W + 9N_{WC} + 4N_{WDC} - 1$ multiplications, $2N_{WC} + N_{WDC}$ divisions, $2N_{WC} + N_{WDC}$ additions, and $N_{WC} + N_{WDC}$ subtractions**. This is dramatically shorter with respect to the number of operations required for computing expression (9) in direct manner, resulting in very fast yield computation.

Note however that, in the ECC-based repair approach using separate CAMs and iterative diagnosis, employing a diagnosis-CAM slightly larger than the runtime-CAM will require very large numbers of iterations, affecting test length adversely. Thus, we use diagnosis-CAMs several times larger than the runtime CAM. From chapter 3 proposition 2, the diagnosis is guaranteed to be successful if the number of *good* CAM locations is larger than the number of memory words affected by two or more faults. As the diagnosis-CAM is much larger than the runtime-CAM (which has sufficient size for repairing all memory words containing two or more faulty cells), the probability that the diagnosis-CAM comprises a number of *good* locations, which is lower than the number of memory words containing two or more faulty cells, is extremely low. Thus, the impact of the diagnosis-CAM on the yield should be extremely low. We have verified this expectation by comparing the results obtained by expression (1) against the results obtained by expression (9), and they are indeed very close. Thus, in the cases of practical interest for the present study (i.e. using diagnosis CAM much larger than the repair CAM), expression (1) can also be used for computing the yield of the separate CAMs architecture.

5.3 Fast Yield Computation for the Overflow CAM Architecture

In this section we derive the analytical expression computing the yield for the Overflow CAM repair architecture, and we also develop mathematics and algorithms for its fast computation. Note also that this expression can also be used for computing the yield for the Overflow Set-Associative Cache repair architecture. Indeed, let N be the number of words of the memory under repair, V1 and K1 be respectively the numbers of sets and of ways of CACHE 1, and V2 and K2 be respectively the number of sets and of ways of the Overflow Set-Associative Cache (CACHE 2). Then, the yield computation for this repair

architecture can be done by considering that we have a system of V2 memories such that: each of the V2 memories comprises N/V2 memory words, and is repaired by means of a set-associative cache having V1/V2 sets of K1 ways each, and an Overflow CAM of K2 locations. Thus, we can use the yield computation approach for the Overflow CAM repair architecture to compute the yield Y of each of the V2 memories. Then, the yield of the memory will be given by $Y_{MEM} = Y^{V2}$.

Let N_S be the number of sets (Set(1), Set(2), … Set(N_S)) of the set-associative cache. As we have seen in section 4.1, the memory is virtually partitioned into N_S memory blocks MB(1), MB(2), … MB(N_S) repaired respectively by Set(1), Set(2), … Set(N_S). Let N_{WS} be the number of locations of each set of the associative-cache (i.e. the number of ways); N_{WB} be the number of words of each virtual memory block; N_d the number of data bits of each memory word (which is also the number of bits of the data field of the set-associative cache and of the Overflow CAM). Let N_{t1} be the number of bits of the tag field of the set-associative cache; N_{t2} be the number of bits of the tag field of the Overflow CAM; and N_f be the number of flag bits in each location of the set-associative cache and of the Overflow CAM.
As we have seen earlier, in ECC-based repair, the probability for a word of the memory to be good (i.e. to contain 0 or 1 faulty cells) is:

$$P_{WMG} = (1 - P_f)^{Nd} + Nd(1 - P_f)^{Nd-1} P_f$$

Also, if the area of the tag cell is r times larger than the area of the SRAM cell and the area of the flag cell is q times larger than the area of the SRAM cell, then, the probability for a location of the set-associative cache to be good is:

$$P_{WSG} = (1 - P_f)^{(rNt1 + qNf)}((1 - P_f)^{Nd} + Nd(1 - P_f)^{Nd-1} P_f).$$

The probability for a location of the Overflow CAM to be good is:

$$P_{WOG} = (1 - P_f)^{(rNt2 + qNf)}((1 - P_f)^{Nd} + Nd(1 - P_f)^{Nd-1} P_f).$$

The probability that a set of the set-associative cache repairs all the faulty words in the corresponding block of the memory (i.e. 0 words are left unrepaired) is:

$$P_{0UF} = \sum_{t=0}^{N_{WS}} \left(\frac{N_{WB}! P_{WMG}^{(N_{WB} - t)}}{(N_{WB} - t)! t!} (1 - P_{WMG})^t \sum_{r=0}^{N_{WS} - t} \frac{N_{WS}! P_{WSG}^{N_{WS} - r}}{(N_{WS} - r)! r!} (1 - P_{WSG})^r \right) \quad \textbf{(2)}$$

The probability that a set of the set-associative cache leaves unrepaired exactly k faulty words in the corresponding block of the memory is:

$$P_{kUF} = \sum_{t=k}^{N_{WS} + k} \left(\frac{N_{WB}! P_{WMG}^{(N_{WB} - t)}}{(N_{WB} - t)! t!} (1 - P_{WMG})^t \frac{N_{WS}! P_{WSG}^{(t-k)}}{(N_{WS} - t + k)! (t - k)!} (1 - P_{WSG})^{(N_{WS} - t + k)} \right) \quad \textbf{(3)}$$

Let = k(1), k(2), … k(N_S) be the number of words of the N_S virtual memory blocks MB(1), MB(2), … MB(N_S) that are left unrepaired respectively by the sets Set(1), Set(2), … Set(N_S) of the set-associative CAM.
The probability that k(1) words in MB1, k(2) words in MB2, … and k(N_S) words in MBN_S are unrepaired is equal to: $P_{k(1)UF} P_{k(2)UF} \ldots P_{k(Ns)UF}$, where the values of the probabilities $P_{k(i)UF}$, are computed by the expression (2) for k(i) = 0 and by expression (3) for k(i) > 0.
In order to repair these faults, the Overflow CAM must dispose at least Q = k(1) + k(2) + … k(N_S) good words.
Let N_{WO} be the number of words of the Overflow CAM. The probability that the Overflow CAM has at least Q fault-free words is:

$$P_{QCO} = \sum_{u=0}^{N_{WO}-Q} \frac{N_{WO}! P_{WOG}^{(Nwo-u)}}{(N_{WO}-u)!u!}(1-P_{WOG})^u \qquad (4)$$

where $P_{WOG} = (1-P_f)^{(rNt2+qNf)}((1-P_f)^{Nd} + Nd(1-P_f)^{Nd-1}P_f)$ is the probability for a location of the Overflow CAM to be good, as determined earlier.

Then, the probability that the memory is repaired when k(1) words in MB1, k(2) words in MB2, … k(N_S) words in MBN_S are unrepaired is:

$PR_{k(1),k(2),\cdots k(Ns)} = P_{QCO}P_{k(1)UF}P_{k(2)UF}\cdots P_{k(Ns)UF}$ **(5)**

where k(1) + k(2) + … k(N_S) = Q, $0 \le k(i)\ \forall\ i \in \{0, 1, \ldots N_S\}$, and P_{QCO} is computed from (4).

For Q = k(1) + k(2) + … k(N_S) > N_{WO} we have $P_{QCO} = 0$. Thus, we only have to consider the cases where Q = k(1) + k(2) + … k(N_S) ≤ N_{WO}. Therefore, to compute the total probability for the memory to be repaired, we have to sum the probabilities $PR_{k(1),k(2),\cdots k(Ns)}$ for all possible value combinations of N_S positive integers k(1), k(2), … k(N_S) having sum equal to Q, for all Q such that $0 \le Q \le N_{WO}$.

In number theory, the combinations of Ns positive integers having sum equal to Q are referred as *the compositions of Q into N_S parts* and is noted as [Q, N_S], also referred as the compositions of length N_S of integer Q). In these compositions 0 is a significant addend (for example 4 + 0 and 0 + 4 are considered distinct compositions of length 2 of integer 4). As an example, the compositions of length 4 of integer 3 are: 3 0 0 0; 2 1 0 0; 2 0 1 0; 2 0 0 1; 1 2 0 0; 1 1 1 0; 1 1 0 1; 1 0 2 0; 1 0 1 1; 1 0 0 2; 0 3 0 0; 0 2 1 0; 0 2 0 1; 0 1 2 0; 0 1 1 1; 0 1 0 2; 0 0 3 0; 0 0 2 1; 0 0 1 2; 0 0 0 3.

The number of compositions of length N_S of integer Q is known to be equal to $C'_{Ns(Q)} = (Q+ N_S-1)!/Q!(N_S -1)!$ which will give huge numbers in many practical cases. For instance, for N_S = 64 and Q = 32, which will be required if we use a set associative cache having 64 sets and 32 ways, we find $C'_{Ns(Q)} \approx 1{,}9801165182011 \times 10^{25}$, which is a huge number of compositions and computing the corresponding probabilities can not be done at reasonable computation time. So, we need to search for solutions accelerating this computation.

We observe that the probability $PR_{k(1),k(2),\cdots k(Ns)} = P_{QCO}P_{k(1)UF}P_{k(2)UF}\cdots P_{k(Ns)UF}$ does not change if we permute the terms of the product $P_{k(1)UF}P_{k(2)UF}\cdots P_{k(Ns)UF}$. Thus, all permutations of a set of integer values k(1), k(2), … k(N_S) will result in the same value for probability $PR_{k(1),k(2),\cdots k(Ns)}$. Therefore, we need to consider only one of the permutations for each set k(1), k(2), … k(N_S). The set of strictly positive integers k(1), k(2), … k(N_S) having sum equal to Q, in which permutations of the same set of integers are considered only once, is known in number theory as *the set of partitions of integer Q*. As an illustrative example, the set of the partitions of 6 is: 6, 5+1, 4+2, 4+1+1, 3+3, 3+2+1, 3+1+1+1, 2+2+2, 2+2+1+1, 2+1+1+1+1, and 1+1+1+1+1+1.

In number theory, the cardinal of this set (i.e. the number of partitions of a positive integer n) is known as the partition function p(n). Leonard Euler's pentagonal number theorem implies

$$p(n) = \sum_r (-1)^{r-1} p(n - r(3r - 1/2)) \qquad (6)$$

where r takes all positive and negative integer values +1, -1, +2, -2, +3, -3, ... The first 14 terms of this relation are p(n) = p(n-1) + p(n-2) – p(n-5) – p(n-7) + p(n-12) + p(n-15) – p(n-22) – p(n-26) + p(n-35) + p(n-40) - p(n-51) - p(n-57) + p(n-70) + p(n -77) - …, where p(0) = 1, p(1) = 1, and p(q) = 0 for q negative. For our experiments, the first 8 terms are sufficient. This is because we use set associative CACHEs with up to 32 ways. Thus we have n = 32. Then, all terms except the first 8 ones are 0. For instance for n = 32 the 9th term is p(32-35) = p(-3) = 0, and so for all terms of order higher than 8. If we use 64-ways set-associative CACHE, the 12 first terms would be sufficient. Indeed the 13th term is p(64-70) = p(-6) = 0, and so for all terms of order higher than 12.

By using the above expression for p(n) we can recursively compute its value for any n: starting from p(0) = 1 and p(1) = 1 we compute p(2), then using the terms computed so far we compute and p(3), and so on. For instance, for n = 32 we find p(32) = 8349, which is dramatically lower than the

$1{,}9801165182011 \times 10^{25}$ compositions of length 64 of number 32. Thus, an algorithm based on the set of partitions of each integer Q in the interval $0 \leq Q \leq N_{WO}$, could accelerate dramatically the computation of the yield for the Overflow CAM repair architecture. As no algorithm generating these partitions is known in the literature, our first step was to develop such an algorithm. This algorithm generates a p(n)xn array PR (where p(n) is computed by means of expression (6) as described above) containing the partitions of the target integer n. To guide the creation of this algorithm we established the following rules:

1) The construction of a partition finishes when the sum of its elements is equal to n.
2) If in partition i-1 (the one generated at step i-1 of the algorithm), the element of position j+1 is larger than 1, then, the jth element of partition i remains the same as the jth element of partition i-1.
3) If in partition i-1 the element of position j+1 is 0 or 1, then, the jth element of partition i is obtained by subtracting 1 from the jth element of partition i-1.
4)
   a) If element j-1 of partition i is produced by rule 3, then, element j will be selected to finish the partition i (i.e. to make the sum of the elements of i to be equal to n, except if this selection contradicts rule 5).
   b) In the latter case, element j will take the value of element j-1, and element j+1 will be selected to finish i, except if this selection contradicts rule 5, and so on.
5) The element in the position j of a partition cannot be larger than any lower order element of the same partition.

Based on these rules the partition generation algorithm is created easily:

% Partitions generation

```
n = Q;    PR = [];
PR = zeros(p(n), n);   % PR is the array in which we store the partitions of n. PR will have
                       % p(n) rows (p(n) = #partitions of n computed by means of expression
                       % (6), as described earlier). Each row of PR will have n elements.
                       % All partitions of n have less than n elements, except partition
                       % 1,1,1, … 1,  which has n elements. Thus, we add the required
                       % number of 0's at each partition to have always a total of n elements.
PR(1, 1) = n;
i = 1;
while PR(i, n) = 0     % By construction, the last partition generated by the algorithm is
                       % 1,1,1 … 1, which is the only partition of n having PR(i, n) ≠ 0.
   i = i + 1;
   j = 1; r = 0;
   while r < n      % Rule 1.
         if PR(i - 1, j + 1) > 1
            PR(i, j) = PR(i-1, j) ; % Rule 2.
            r = r + PR(i , j);
            j = j + 1;
      end
      else
         PR(i , j) = PR(i-1, j) - 1; % Rule 3.
         r = r + PR(i , j);
         j = j + 1;
         while r < n
            if n - r > PR(i , j-1)
               PR(i , j) = PR(i , j-1) ; % Rule 4.b.
            end
            else
```

```
                PR(i , j) = n – r; % Rule 4.a.
            end
            r = r + PR(i , j);
            j = j + 1;
        end
    end

  end
end
```

We will use this algorithm to generate the p(Q)xQ array PR, where Q takes all integer values 0, 1, 2, … N_{WO}. However, if $N_{WO} > N_S$ (where N_{WO} and N_S are respectively the number of locations of the Overflow CAM and the number of sets of the set-associative cache), then certain values of Q will be larger than N_S. As the rows of PR have Q elements, they will provide partitions of Q into a number of integers larger than N_S, but we are looking for partitions into N_S integers k(1), k(2), … k(N_S). Thus, the above algorithm generates the required arrays PR for each $Q \leq N_S$. On the other hand, for each $Q > N_S$, we transform PR by eliminating its rows, which contain more than N_S non-zero elements. In the reduced array, we eliminate all columns of order higher than N_S (in which, by the way, all elements are equal to 0). As the algorithm realizing this transformation is trivial, it is not presented it here.

Now we can use the array PR to compute the yield for the Overflow CAM repair architecture by means of the following algorithm:

% Yield computation for the overflow CAM repair architecture

```
PR = []; CP = []; CP = zero's(p(Q), Q+1); NP = []; PKQ =;
```
Yield = $P_{0UF} \wedge Ns$; %In fact this initialization is $P_{0CO} * P_{0UF} \wedge Ns$, but from (3) we find $P_{0CO} = 1$.
```
for  Q = 1 : N_WO
   PKQ = 0;
   Compute P_QCO        %From equation 3
   Generate array PR   %By calling the subroutine "Partitions generation" described above
   for  i = 1 : p(Q)   % Generation of an array CP, which counts the number of occurrences in
            % the row i of array PR of each integer j, 0 ≤ j ≤ Q.
        for j = 1 : Q
            h= 1;
            while PR(i, h) > 0
                  if PR(i, h) = =  j
                        CP(i, j) = CP(i, j) + 1;
                  end
                  h = h+1;
            end
                    CP(i, Q+1) = N_S – h +1;   %CP(i, Q+1) computes the number of 0's in row
                                % i of PR.
        end
   end
   for i = 1 : p(Q)        % Compute PR_k(1),k(2),…k(Ns) (using equation (4)) for each row i
                  % of PR; Use array CP to compute the number of permutations of the
                        % partition stored in row i of PR[16]. Store the results in row i of NP.
```

[16] As each row of PR has N_S elements, there are N_S! possible permutations. But, as several of these elements may take the same value (i.e. may have multiplicity higher than 1), several of the N_S! permutations will be identical. To eliminate them, we divide N_S! by the factorial of this multiplicity.

```
        NP(i, 1) = P_QCO ; NP(i, 2) = factorial(N_S);
        for j = 1 : N_S
            k= PR(i, j);
            compute P_kUF        %from equation 1 if k = 0, from equation 2 otherwise.
            NP(i, 1) = NP(i, 1)P_kUF;
            If  j < = Q+1
                    NP(i, 2) = NP(i, 2)/factorial(CP(i, j));
            end
        end
        PKQ = PKQ+NP(i, 1)NP(i, 2);    %Compute the probability that Q memory words
                                       % are left unrepaired by the set-associative
                                       % CACHE and are repaired by the overflow CAM.
    end
    Yield = Yield + PKQ;
end
```

For easier understanding of this algorithm the following explanation can be useful.

The algorithm explained

i. For each integer Q of interest (i.e. in the interval $0 \le Q \le N_{WO}$), we determine its set of partitions. For instance, for Q = 3, we obtain the following partitions: {3}; {2, 1}; {1, 1, 1}. Each of these partitions has less than Q elements, except the partition {1, 1, … 1}, which has Q elements. We add to each partition the necessary number of 0's in order to obtain a row of N_S elements, and store it in a row of an array PR of size p(Q) x N_S. The algorithm realizing the set of partitions of an integer Q is presented later.

ii. We create an array CP of size p(Q) x (Q+1). The element CP(i, j) of this array is computed by counting the number of times the value Q – j +1 appears in row i of PR. For instance, if row i of PR is 2, 1, 0, 0, then, 3 appears 0 times, 2 appears 1 time, 1 appears 1 time and 0 appears 2 times and the row i of CP will be equal to 0, 1, 1, 2.

iii. For each row i of the array PR we compute the probability $PR_{k(1),k(2),\dots k(Ns)} = P_{k(1)UF}P_{k(2)UF}\dots P_{k(Ns)UF}P_{QCO}$ by setting k(1) = PR(i, 1), k(2) = PR(i, 2), … k(N_S) = PR(i, N_S). We store these probabilities to the first column of an array NP.

iv. We compute the number of permutations for row i of PR as (N_S!)/(CP(i, 1)!(CP(i, 2)! … CP(i, Q+1)!). Indeed, as each row of PR has N_S elements, the number of permutations the elements of the row is N_S!. However several of these N_S elements take the same value. This implies that several of the N_S! permutations are identical and have to be eliminated. Thus, we have to divide N_S! by the factorial of the number of times each value appears in the row i of PR. This number is given by the elements of the ith row of array CP. For instance, in the example used in step ii, the row i of CP is 0, 1, 1, 2. Thus, the number of permutations is 4!/(0!1!1!2!) = 12. We store the numbers of permutations in the second column of array NP.

v. The probability that the memory is repaired when Q memory words are left unrepaired by the set associative CACHE is computed as $\sum_{i=1}^{p(K)} NP[i,1]NP[i,2]$.

vi. We repeat the above for all Q such that $0 \le Q \le N_{WO}$. Each time we obtain a probability from step v. We add the resulting N_{WO} + 1 probability, to obtain the total probability to have the memory repaired by the combination of the associative CACHE and of the overflow CAM.

As discussed earlier the above algorithm reduces dramatically the number of products that have to be computed in order to determine the yield, reducing dramatically the computation complexity. The last sources of computation complexity are the time-consuming computations of the terms P_{QCO} and $P_{k(1)UF}$, $P_{k(2)UF}$, … $P_{k(Ns)UF}$, used in the product $PR_{k(1),k(2),\ldots k(Ns)} = P_{QCO}P_{k(1)UF}P_{k(2)UF}\ldots P_{k(Ns)UF}$.
To accelerate the computation of these terms, we have derived recursive relations, which reduce drastically the number of the required operations:

$P_{QCO} = \sum_{u=0}^{N_{WO}-Q} \frac{N_{WO}! P_{WOG}^{(N_{WO}-u)}}{(N_{WO}-u)!u!}(1-P_{WOG})^u$ can be written recursively as:

$$P_{QCO} = \sum_{u=0}^{N_{WO}-Q} X_u \qquad X_0 = P_{WOG}^{N_{WO}},\ X_{u+1} = \frac{N_{WO}-u}{u+1}\frac{(1-P_{WOG})}{P_{WOG}}X_u$$

These relations allow computation of each term computing each term P_{QCO} at linear time. Concerning the terms P_{kUF}, we need to compute them for all integers k in the interval $0 \le k \le N_{WO}$. For k = 0, P_{0UF} is given by expression (2), which is identical to the expression (1) in section 5.1, and can be computed fast by using the recursive relations derived in section 5.1 for expression (1).

For k > 0 we use expression (3) giving:

$$P_{kUF} = \sum_{t=k}^{N_{WS}+k} \frac{N_{WB}! P_{WMG}^{(N_{WB}-t)}}{(N_{WB}-t)!\,t!}(1-P_{WMG})^t \frac{N_{WS}! P_{WSG}^{(t-k)}}{(N_{WS}-t+k)!\,(t-k)!}(1-P_{WSG})^{N_{WS}-t+k}$$

Let us set:

$$P_{kUF} = \sum_{t=k}^{N_{WS}+k} Y_t^k$$

Then, for $t \ge k$ Y_t^k can be written recursively as:

$$Y_k^k = \frac{N_{WB}! P_{WMG}^{(N_{WB}-k)}}{(N_{WB}-k)!k!}(1-P_{WMG})^k (1-P_{WSG})^{N_{WS}}$$

$$Y_{t+1}^k = Y_t^k \frac{(N_{WB}-t)}{(t+1)}\frac{(1-P_{WMG})}{P_{WMG}}\frac{(N_{WS}-t+k)}{(t-k+1)}\frac{P_{WSG}}{(1-P_{WSG})}$$

From these relations, P_{kUF} can be computing by the following fast algorithm.

%Fast computation of P_{kUF}

$Y_t^k = \frac{N_{WB}! P_{WMG}^{(N_{WB}-k)}}{(N_{WB}-k)!k!}(1-P_{WMG})^k (1-P_{WSG})^{N_{WS}};$ % Computation of Y_k^k

$P_{kUF} = Y_t^k;$

for t = k+1 : N_{WS}+ k

r = t-1;

$Y_t^k = Y_t^k \frac{(N_{WB}-r)}{(r+1)}\frac{(1-P_{WMG})}{P_{WMG}}\frac{(N_{WS}-r+k)}{(r-k+1)}\frac{P_{WSG}}{(1-P_{WSG})};$

$P_{kUF} = P_{kUF} + Y_t^k;$

end

Using these fast computation algorithms for P_{QCO}, P_{0UF}, P_{kUF} our fast yield-computation algorithm for the Overflow CAM repair scheme is further accelerating, enabling computing in short time the yield for the sophisticated Overflow CAM and Overflow Set-Associative Cache repair architectures.

5.4 CONCLUSION

In high defect densities the analytical computation of the yield achieved by a repair architecture becomes increasingly complex because: while in low defect densities the faults affecting the replacement units (e.g. the repair CAM) have negligible impact on the yield and can be neglected in the yield computation, in high defect densities this impact is significant and requires using more complex yield computation expressions. The numbers of operations required for computing these expressions are: $Nw(Nwc + 1) + (Nwc^2 - 1)(5Nwc + 12)/6 - 1$ high precision multiplications; $(Nwc + 1)(Nwc + 4)/2$ high precision divisions; and $N_{WC}(N_{WC} + 3)/2$ additions, where Nw is the number of the words of the memory under repair and Nwc is the number of locations of the repair CAM. Thus, the number of these operations increases exponentially with the size of the memory under repair and of the repair CAM. This exponential increase makes the computation of these expressions intractable, because in the technologies targeted by the present study we have to consider very large memories under repair, as well as very large repair CAMs. Indeed, on the one hand, ultimate CMOS and post-CMOS technologies will enable integrating in the future much larger memories than in nowadays technologies, and on the other hand, the repair CAM becomes very large due to the large size of the memory under repair and the high defect densities. To cope with these issues, in this chapter we discovered several recursive relations enabling computing the analytical expressions of the yield at linear time ($N_W + 8N_{WC} - 1$ multiplications, $2N_{WC}$ divisions, $2N_{WC}$ additions, and N_{WC} subtractions). This original results lead to a dramatic reduction of the yield computation complexity, making it possible even for huge memories and very high defect densities.

Yet another yield computation issue arises with the introduction of sophisticated memory repair architectures as the ones presented in chapter 4. The complex analytical expressions that we have derived for computing the yield achieved by these architectures, require huge numbers of operations. For instance, if we use a repair cache having 64 sets and 32 ways, we need to compute $1{,}98x10^{25}$ very complex terms, requiring huge computation time. Thanks to several mathematical developments presented in section 5.3, we succeeded to reduce these operations by more than 20 orders of magnitude, resulting in very fast yield computation even for the most sophisticated repair architectures.

These developments can be easily adapted to other sophisticated architectures using hybrid repair schemes. Thus, they represent significant achievements, enabling fast and high precision yield computation for future technologies and the sophisticated hybrid repair architectures that should be used for coping with the increased defect densities.

CHAPTER 6

TRANSPARENT BIST FOR ECC-BASED MEMORY REPAIR

As highlighted in chapter 1, ECC-based repair drastically reduces area and power penalties with respect to conventional repair, but these advantages can be lost due to diagnosis requirements, which may lead in similar hardware cost as for conventional repair. In the previous chapters we developed a compendium of approaches enabling resolving these issues, including: an ECC-based repair scheme using separate diagnosis CAM and repair CAM, which reduces the runtime power dissipation with respect to conventional repair; a new family of test algorithms completely eliminating the diagnosis hardware, hence reducing drastically both area and power penalties with respect to conventional repair; an iterative diagnosis approach enabling trade-offs between area penalty and test length; as well as, new fast simulation and yield computation methods enabling evaluating these approaches in short computation time. Though these schemes drastically reduce area and power penalties, further power reduction is welcome. Thus, the above approaches were completed by new word repair architectures, which allow further reduction of power dissipation, together with new yield computation mathematics enabling fast evaluation of these architectures. Hence, the developments in chapters 1, 2, 3, and 4, result in a comprehensive framework allowing efficient memory repair for high defect densities.

These developments consider that fault diagnosis required for performing ECC-based repair uses conventional memory BIST. However, circuit aging is drastically accelerated as we approach ultimate CMOS and post CMOS. In combination with aggressive voltage reduction, it will result in very frequent occurrences of faults during application execution. This imposes testing the memories frequently during application execution to discover such faults before the system commits faulty results. Testing the memories during application execution often requires preserving their content. Transparent BIST, introduced in [40][41] and further developed and used by numerous authors and fault-tolerant systems [42-49], transforms memory test algorithms into reversible processes, which preserve the memory content. However, *while test data verification in conventional memory BIST uses test data comparison, test data verification in transparent BIST uses signature analysis*. Unfortunately, signature analysis-based test data verification is not compatible with ECC-based repair. Thus, in this chapter, a hybrid diagnosis scheme combining signature analysis with ECC error detection and correction is proposed, allowing smooth cooperation between transparent BIST and ECC-based repair.

6.1 TRANSPARENT BIST VERSUS ECC-BASED REPAIR: ISSUES AND COOPERATION STRATEGY

ECC-based repair admits words containing a single faulty cell and repairs words comprising multiple faulty cells. Thus, fault diagnosis should ignore words comprising a single faulty cell and locate words comprising multiple faulty cells.

The work presented in this chapter, as well as the work presented in the previous chapters, are developed in the context of *Cells* framework [23][25][50]. This framework supports the design of massively parallel tera-device processors affected by high defect densities. *Cells* employs three error recovery approaches: instruction replay; error-recovery based on coordinated checkpoint [51]; or checkpoint-free error recovery that exploits a parent-child hierarchy of the software tasks, to re-execute aborted child tasks [27]. Instruction replay is used only for recovering timing and transient faults in logic parts. Thus, for memory faults, only the latter two error-recovery approaches are of interest.

Cells, activates error recovery for memory faults in three cases: ECC detects uncorrectable error during application execution; a memory self-test detects a memory word comprising multiple faulty cells, a CAM self-test detects a faulty tag field in the run-time CAM (described later) or a data field of this CAM comprising multiple faulty cells.

The memory self-tests are executed in a manner that avoids the contamination of the system by errors induced when new memory faults occur during application execution. For this purpose, in the checkpoint-based error recovery approach, self-test is executed before the old checkpoint is aborted and new checkpoint is taken. In the checkpoint-free error recovery approach, self-test is executed before the results of certain child tasks are committed by their parent task and are further distributed to the system. In both cases, the goal of the self-test is to determine if the memory contains words comprising multiple faulty-cells that could have produced errors undetectable by the ECC, in order to:

- Initiate preventive error recovery (using fault-tolerant task scheduling and allocation for redistributing the aborted tasks to fault-free resources of the processor array (see [27])), before unrecoverable error contamination of the system.
- Launch a repair session to replace by spares the words comprising multiple faulty cells. Thus, the system can again use the repaired memory.

As we describe later, to reduce test time, self-tests are executed only in physical blocks of the memory that were used after the last checkpoint (first error recovery approach), or during the execution of child tasks whose correctness has to be checked (second recovery approach).

Also, as these tests are performed during application execution, memory blocks may contain useful information. Thus, transparent BIST will be used to preserve their contents.

The difficulty is that transparent BIST uses as test data the memory contents transformed by reversible operations [40]. As these data are unknown, they cannot be verified by comparing the data read from each memory word against their known fault-free value. Instead signature analysis is used, based on a signature prediction scheme [40]. Signature analysis is convenient when we only need to know if the memory is faulty or fault-free (go-no go test). But, *it cannot distinguish erroneous responses coming from words containing single-errors, from erroneous responses coming from words that may contain multiple errors*. Thus, it is not suitable for ECC-based repair. Then, instead of verifying test data by means of signature analysis, another option would be to use the ECC for checking the result of each read operation. However, this approach may fail for memory words producing multiple errors undetectable by the ECC (i.e. a triple error for the SECDEC codes). To cope with this issue we propose a hybrid architecture combining signature analysis and ECC, enabling determining if the memory contains words that comprise multiple faulty cells, as described in the next section. To make it possible we relax the ECC-based diagnosis requirements during transparent BIST, by adopting a two-step diagnosis process:

- First step: This step uses transparent BIST to avoid destroying the memory contents. It determines if the memory contains not yet repaired words that comprise multiple faulty cells (due to faults occurred after the previous self-test), but does not determine the number and location of such words. In the most frequent case, where no such words are discovered the second step is skipped.
- Second step: This step is executed in the less frequent case where the first step has discovered that the memory contains unrepaired words comprising multiple faulty cells. In this case, the application tasks executed over the memory blocks that contain these words are aborted and are allocated for re-execution to other fault-free resources of the multiprocessor array. Thus, the contents of the faulty memory can be discarded and the memory is tested with conventional (i.e. non-transparent BIST), which allows to

identify the word(s) that comprise multiple faulty cells and to repair them.
Based on the above strategy, the proposed transparent BIST architecture is described in the next section.

6.2 TRANSPARENT BIST FOR ECC-BASED REPAIR

ECC-based repair uses a CAM for storing words comprising multiple faulty cells (run-time CAM). This CAM is much smaller than the CAM used for conventional word repair and induces small area overhead. Its power dissipation is also drastically reduced, but it was still not negligible. Thus, a new word repair architecture was introduced in chapter 4, which drastically reduces this power. Diagnosis for ECC-based repair may require a large diagnosis CAM, which will induce large area penalty and also large power dissipation during the test session. To cope with this issue, two approaches were proposed in chapters 2 and 3. The first introduces a new family of memory test algorithms (SRDF test algorithms), which eliminates the diagnosis CAM at the expense of test time. The second uses an iterative diagnosis scheme that enables trading test time with diagnosis CAM size.
In this chapter, we propose two transparent BIST architectures. The first one targets the approach using SRDF test algorithms. The second, rather than using the iterative diagnosis approach (which reduces the size of the diagnosis CAM at the expense of extra test time), uses a memory-block based diagnosis approach introduced in the next section, which reduces both the size of the diagnosis CAM and the test time.

6.2.1 Block-based Test and Diagnosis Strategy

A memory array is usually organized into rows and columns. A row contains several memory words and is selected by a row line activated through the decoding of a group of address bits (row address bits). A second group of address bits (column address bits) is used to select the target word among the words of the selected row. Large rows and columns have large capacitances whose charging and discharging induce large power dissipation and signal delays. Thus, to reduce power and increase speed, word-line splitting and bit-line splitting is often used. Furthermore, memories are partitioned into several physical blocks (banks) selected by a group of address bits (bank-address bits). This partition reduces the size of rows and columns within each bank, reducing delay and power dissipation. In particular, as one bank is selected at each memory access, power is reducing drastically. We exploit this organization in order to reduce the time duration of self-tests, as well as the size of the diagnosis CAM in the repair approach using separate diagnosis and runtime CAMs proposed in chapter 3.

Block-test scheme: For each memory we use a register (bank identification register – BIR) having a number of bits equal to the number of the memory banks. These bits identify the memory banks that were used between two consecutive self-tests of the RAM. The identification is performed by decoding the bank-address bits to generate the values stored in BIR. For instance, for a memory partitioned into 32 physical banks, BIR will have 32 bits. At the end of each self-test session, BIR is reset to the all 0's state. Then, during the application execution, at each memory access the 5 bank-address bits are decoded to generate a 1-out-of-32 code. The 32 bits of this code are bit-wise ORed with the contents of BIR to generate its new contents. During the next self-test, we sequentially read the bits of BIR. Each time a bit is equal to 1, we activate the self test of the corresponding memory bank. Thus, the tested banks are those that were used by the application in the interval separating the previous self-test phase from the current self-test phase.
As the untested memory banks were not used since the last self-test, then, there is no risk that a word belonging a non tested bank and containing multiple faulty cells could contaminate the system with undetectable errors. Nevertheless, to avoid accumulating multiple faults in the words of RAM banks that could be left untested for too long (i.e. banks that are not used for long time by the application), these tests are complemented by self-tests, which test the banks that remain untested for more time than a given threshold.

Block-based diagnosis scheme: In this scheme we use a common diagnosis CAM for all banks of a memory. This CAM is sized to accommodate all faulty words of the memory that comprise multiple faulty cells, as well as the faulty words that comprise single faulty cells of one memory bank at a time. Then the test and diagnosis process works in the following manner. We test each memory bank as described above, and we store in the diagnosis CAM all faulty words detected in this bank (i.e. we store the address of the faulty word in the tag field of a good location of the diagnosis CAM[17] and the faulty-cell positions of this word in the associated data field). At the end of this test, we clear all CAM locations that contain words comprising only one faulty cell, and we conserve the words comprising multiple faulty cells. As the large majority of faulty memory words will contain only one faulty cell, this approach liberates the majority of the diagnosis CAM locations, which can be used for diagnosing the next tested bank of the memory, and so on. Thus, the size of the diagnosis CAM is reduced significantly. For instance, for a defect density 10^{-3} (i.e. 1 out of 1000 memory cells is faulty), and 32-bit memory words, the probability for a word to contain one faulty cell is $P_{w1f} = n(1-Pf)^{n-1}Pf = 0{,}03102274$ (with $n = 32$ and $Pf = 10^{-3}$), while the probability for a memory word to contain more than one faulty cell is $P_{wmf} = 1- (1-Pf)^n - n(1-Pf)^{n-1}Pf = 0{,}00048618$. Thus, in a memory comprising 100K words of 32 bits, the mean number of words containing one faulty cell will be equal to 3102 and the mean number of words containing more than one faulty cell will be equal to 48. Thus, if we do not use the block-test and repair approach, the size of the diagnosis CAM will be roughly equal to 3150 words (3102 + 48). More detailed probabilistic computations are required for determining the size of the diagnosis CAM for a target success probability, which generally will be larger than 3150 words if we target high success probability. For instance, if we require a 95% probability for the diagnosis to be successful, we find that the size of the diagnosis CAM should be 3535 words. On the other hand, if the memory is partitioned into 32 banks, the mean number of words in a bank comprising one faulty cell will be equal to 98. Thus, if we apply the block-test and diagnosis scheme we will need a diagnosis CAM of size roughly equal to 146 (98 + 48). This size will be larger if we target high success probability. For instance for 95% success probability we will need a diagnosis CAM of 186 words. Thus, the bank test and diagnosis scheme results in drastic reduction of the diagnosis CAM size (form 3535 down to 186 CAM locations in the above example).

Let us now compare the block-based diagnosis scheme with the diagnosis approaches presented in chapters 2 and 3:

- With respect to the diagnosis scheme using SRDF algorithms, the block-based diagnosis scheme induces higher area penalty. For instance, in the above example, the SRDF algorithms-based diagnosis will employ a runtime CAM of 48 locations, while the block-based diagnosis scheme uses a runtime CAM of the same size but also a diagnosis CAM of 146 locations. However, as the block-based diagnosis scheme uses conventional test algorithms, it allows a significant reduction of test time with respect to the test time required by the SRDF test algorithms.
- With respect to the diagnosis scheme using separate diagnosis CAM, the block-based diagnosis scheme induces drastically lower area penalty. For instance, in the above example, the diagnosis scheme using separate diagnosis CAM requires 3150 diagnosis-CAM locations, while the block-based diagnosis scheme requires only 146 locations. Also, as both schemes employ conventional test algorithms, they require the same test length
- With respect to the scheme combining separate diagnosis CAM with iterative diagnosis, the block-based diagnosis scheme allows significant reduction of the size of the diagnosis CAM without increasing test length.

Thus, the block-based test and diagnosis scheme is a promising concept for performing advantageous trade-offs in terms of area penalty and test time, whose evaluation and validation was not yet realized due to time constraints, and will represent in our future developments a promising extension of the work

[17] Good CAM locations are identified by dedicated flags[19].

presented in this manuscript.

Further to these merits, in the context of self-tests performed during application execution in order to avoid that undetected memory faults are propagated in the system, the block-based test using the Bank-Identification-Register (BIR), allows significant reduction of the self-test duration.

These advantages become possible because, as shown in the following proposition, the block-test and diagnosis scheme preserves the quality of the test and repair process.

Proposition 1: The block-test and diagnosis scheme does not affect test and diagnosis quality.

Proof: As the test algorithm is executed separately for each bank of the memory, then, if a fault affecting a cell that belongs to a memory bank is sensitized by operations performed over cells belonging to different banks, the block-test approach will mask this fault. However: cells belonging to different memory banks are topologically very distant; they do not share any signal (e.g. word-lines or bit-lines), nor read or write amplifiers; and they are activated at different times (only one bank of a memory can be accessed at a time). Thus, couplings between cells belonging to different memory banks cannot occur. Hence, a faulty cell belonging to a bank cannot be sensitized by operations performed over cells belonging to other banks. Hence, the block-test scheme does not mask any fault. This also implies that: resetting at the end of the test of a memory bank the locations of the diagnosis CAM storing one faulty-cell position (as described in the block-based diagnosis scheme), does not compromise the diagnosis of words containing multiple faulty cells. Indeed, as all the faulty cells of a bank are detected during the test of this bank, there is not risk that during the test of another bank we discover new faulty-cells affecting a word stored in diagnosis CAM. Hence, the locations of the diagnosis CAM, which store at the end of the test of a memory bank one faulty-cell position, concern words containing only one faulty cell. Thus, resetting them at the end of the test of a memory bank does not compromise the diagnosis of words containing multiple faulty cells. **QED**

6.2.2 Transparent BIST Architecture for ECC-based Repair

As mentioned in section 6.1, in the context of ECC-based repair we should be able to determine if the memory contains words that comprise two or more faulty cells. However, *transparent BIST is not able to distinguish the case where all faulty words comprise single faulty cells from the case where some of them comprise multiple faulty cells*. This is because *in contrast with conventional memory BIST, which compares read data against their expected values, transparent BIST verifies read data by means of signature analysis*. In addition, *using the ECC code to verify the read data cannot distinguish words containing single errors from words containing certain multiple errors. It may also not at all detect certain multiple faults*. This may make transparent BIST incompatible with ECC-based repair. To resolve this issue we introduce a new test response verification and diagnosis architecture merging signature and ECC checks.

This architecture works as described below:

i. During self-test the repair circuitry is active. Previously repaired words (i.e. replaced by spares) are not accessed by the test algorithm. Thus, faults in already repaired words do not produce errors during self-tests.
ii. The read data are injected into the signature analyzer after correction by the ECC code.
iii. The *diagnosis CAM stores the error syndromes produced by the ECC circuitry*. To do this, each CAM location will comprise a tag field used to store the address of a faulty memory word, and a data field, which, instead of storing the positions of the faulty cells of this word discovered by a conventional BIST circuitry, will store the error syndrome produced by the ECC. Then, the diagnosis CAM will be updated in the following manner. Each time the ECC circuitry detects an error, the current memory address is compared in parallel with the contents of all tag fields. In case of miss, the current address is stored in an unoccupied tag field and the error syndrome is stored in the corresponding data field. In case of hit, the data field of the hit CAM location is read; its content is bit-wise ORed with the current error syndrome; and the result of the bit-wise OR operation is written back in the data field of the hit CAM location. This

way, all positions of errors detected in the same memory word during the read operations of the test algorithm are accumulated in the data field of a single CAM location. Thus, reading the data fields of the CAM at the end of the test phase allows discovering the cumulated syndromes of the errors detected in each memory word.

iv. If at the end of the test phase the signature is erroneous and/or if some data field(s) of the diagnosis CAM contain 1's in multiple positions, and/or if at any time during the execution of the test algorithm the ECC has indicated the detection of a double error, then, the diagnosis concludes that some memory word(s) comprise multiple faulty cells. In all other cases the diagnosis concludes that no memory word comprises multiple faulty cells.

Proposition 2. The test response verification and diagnosis architecture described above can determine if the memory contains words comprising multiple faulty cells, provided that these faulty cells are detected by the transparent test algorithm.

Proof. We consider single-error correcting, double-error detecting (SECDED) codes (e.g. Hamming or Hsiao). More complex codes offering higher error correction and detection capabilities will have higher diagnosis capabilities. Thus, the proof will also be valid for such codes.

Note that, since all faulty cells are detected by the transparent test algorithm, each faulty cell will produce an error in some read operation of this algorithm. Also, if a word contains more than one faulty cell, it is possible that the detection of these faulty cells may not occur in the same read. Thus, a word comprising multiple faulty cells may produce only single errors in each read operation of the test algorithm.

All possible situations concerning error multiplicity affecting the data read during the test algorithm are:

a. No read contains more than one error.

b. Some reads contain two errors.

c. Some reads contain more than two errors and no read contains two errors.

In case *a*, each data word injected to the signature analyzer is correct, as the ECC corrects single errors. Thus, the signature does not detect any fault. On the other hand, as the errors affecting any read word are always single, the error syndromes correctly identify the positions of these errors, which are stored in the diagnosis CAM. We have also seen that for each faulty cell the test algorithm produces an error in some read operation. Thus, the error syndromes will provide the correct position for each faulty cell, which will be stored in the diagnosis CAM. Then:

- If no memory word contains more than one faulty cell, then the content of the diagnosis CAM will indicate at most one faulty cell per memory word. Combined with the fact that the signature is correct, according to the diagnosis decisions described above in point iv, we will diagnose that no memory word contains multiple faulty cells, which is correct.
- If some memory word contains multiple faulty cells, then, the position of each of these cells will be indicated in the error syndromes of the ECC, which are cumulated (by XORing them) in the data field of a locations of the diagnosis CAM, and according to point iv we will diagnose that some memory word contains multiple faulty cells, which is correct.

Case *b* implies that a memory word comprises at least two faulty cells (the words where some reads detect double errors). Also, since some read(s) contain two errors, then, the ECC will detect the double error, and according to point iv we diagnose that some memory word contains multiple faulty cells, which is correct. Also, as the code cannot correct the double error, the signature analyzer will receive erroneous data and will detect an error indicating that some memory word(s) contains multiple faulty cells, and according to point iv we diagnose that some memory word contains multiple faulty cells, which is correct. Thus, in case b two criteria announced in point iv diagnose correctly that some word(s) comprise multiple faults.

Case *c* implies that a memory word comprises more than two faulty cells (in the positions where the read contains errors of multiplicity larger than two). As the ECC corrects single errors and detects double errors, then, during a read containing more than two errors, the ECC circuitry may: incorrectly indicate that the word does not contain any error; or indicate that the word contains uncorrectable errors; or incorrectly

indicate that the word contains a single error and produce an error syndrome wrongly indicating a single error position. In all cases, incorrect data are injected to the signature analyzer, which provides an incorrect signature. Then, thanks to the wrong signature, according to point iv we diagnose that some memory word(s) contains multiple faulty cells, which is correct. **QED**

Note that, as stated in section 6.1, if the transparent test discovers that some memory words contain multiple faulty cells, then, a non-transparent test is executed over the concerned memory bank, in order to precisely locate these words and to replace them by spare ones. Nevertheless, from the above analysis we observe that the execution of non-transparent test is required only when the signature is erroneous (cases b and c). When the signature is correct (case a), the diagnosis CAM correctly identifies these words, and the repair is done by the transparent test. Thus, *non-transparent test will be executed only in the infrequent (but non negligible) situation where, since the last time the memory bank was tested, two new faults have occurred in a memory word already containing one faulty cell, or more than two faults have occurred in any memory word*.

Note also that, as we address high defect densities, a large percentage of memory words will contain one faulty cell. For instance, considering memory words of 32 data-bits plus 7 ECC check bits and a $3x10^{-4}$ defect density, then, roughly 1.17% of memory words will contain one faulty cell. For a 10^{-3} defect density roughly 3.9% of the memory words will contain one faulty cell. In addition, the combination of accelerated aging with aggressive voltage reduction would drastically increase the rate of new faulty cells occurring in the field. Thus, the probability that between two test sessions some words may accumulate more than two faulty cells (e.g. two new faulty cells occur in a word already containing one faulty cell) is low but not negligible. This, assumption leads to the error cases described earlier in points *a*, *b* and *c*, which require the above described hybrid diagnosis scheme combining ECC-based error detection with signature analysis. If we consider moderate defect densities and aging rates, the probability that a memory word accumulates more than two faulty cells in the interval separating two test sessions could become negligible. Then, the error case described earlier in point *c* can be neglected. In this case, the diagnosis can be performed only by using error detections based on the ECC code, as described bellow:

In case *a*, as the errors affecting any read are always single, the error syndromes correctly identify their positions. These positions are stored in the diagnosis CAM. Thus, the contents of this CAM will allow correctly diagnose the existence or absence of words containing double faulty cells.

In case *b*, if a read operation contains a double error, the ECC will signal it and we will correctly diagnose that some memory word(s) contain double faulty-cells.

Note that, this simplified diagnosis scheme eliminates the signature analysis and requires slightly less complex FSM for controlling the diagnosis process. However, the complexity of both circuits is low, while the more complex blocks (diagnosis CAM and run-time CAM) are identical in both diagnosis schemes. Thus, the cost reduction will be low and in most cases may not justify the lost in diagnosis capabilities implied by the simplified diagnosis scheme.

6.2.3 Fault Coverage, Transparent Test Algorithm, Signature Prediction, and ECC-Consistency

Fault Coverage and Signature Prediction: Proposition 2 stipulates that the proposed transparent BIST architecture for ECC-based repair, using block-based test and diagnosis, determines if the memory contains words comprising multiple faulty cells, provided that these faults are detectable by the transparent test algorithm. The detection of faults is fundamentally determined by the memory test algorithm, which is used to derive the transparent test algorithm. As an illustration we will consider the set of all static unlinked functional fault models (FFMs) involving one memory cell (single-cell FFMs) and two memory cells (two-cell FFMs) [29], and the optimal march test algorithm MSS1 [31] detecting all of them. Algorithm MSS1 is presented in figure 1. It is composed of a march element S(0) used for initializing the memory to a known state, and five march elements S(1) to S(5) used for testing the memory.

The transparent test algorithm derived from MSS1 according to the rules proposed in [40] is presented

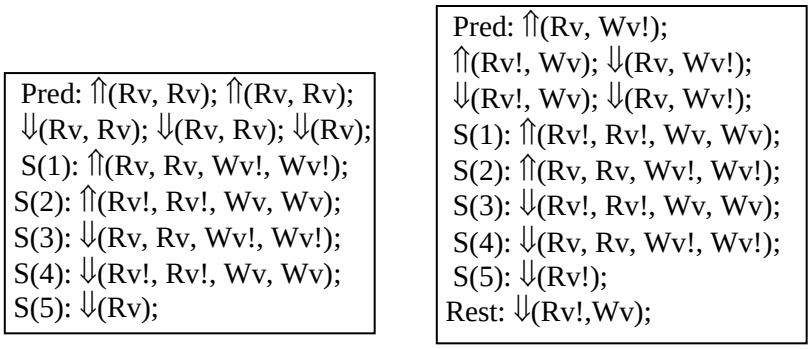
in figure 2.a. In the five march elements S(1) to S(5) of this algorithm the value 0 is replaced by v (v does not take a fixed value but varies from a memory word to another, taking the initial value of each word), and the value 1 is replaced by v! (representing the inverse of v). When each of the march elements S(1) to S(5) visits a memory address, the value v or v! is read from this address. This value is stored in a register (the transparent-test data register) and is used to produce the value v! or v employed by the subsequent writes performed over this address. As the contents of the memory are unknown, the signature produced by the march elements S(1) to S(5) is also unknown. Thus, the algorithm starts with a signature prediction sequence, labelled as Pred in figure 2. It consists in five march elements obtained by removing the write operations from march elements S(1) to S(5) of the transparent test algorithm. However, as Pred is composed only from read operations, the data v will appear in all instances of Pred. Then, in order for the fault-free operation to produce the same signature as march elements S(1) to S(5) (signature prediction), the data read in the second and fourth march elements of Pred are inverted before being injected in the signature analyzer.

Concerning fault coverage, if the fault model is complete (i.e. for each fault type the model comprises all faults corresponding to all possible state combinations of the victim and aggressor cells), then, the transparent test algorithm provides the same fault coverage as the conventional test algorithm from which is derived [40]. Thus, as MSS1 covers all static unlinked single-cell and two-cell FFMs [29], which is a complete fault model (each FFM type comprises faults corresponding to all possible value combinations of the victim and aggressor cells [29]), the transparent test algorithm of figure 2.a will also detect all static unlinked single-cell and two-cell FFMs. Nevertheless, masking could occur if some faults produce the same errors in the signature prediction sequence (Pred) and in the test march elements (S(1) to S(5)). As Pred includes only read operations, this could happen only for faults sensitized by reads [40]. As the single-cell and two-cell FFMs include such faults [29], fault masking becomes possible. To exclude this possibility, we propose a new kind of signature prediction shown in figure 2.b.

In figure 2.b, the march elements of sequence Pred include only one read operation, while march elements S(1) to S(4) include two read operations. Thus, to correctly predict the signature, during the first, second, third and fourth march elements of Pred, the value of the read operation is injected to the signature analyser and is also stored in a register (the transparent test data register mentioned earlier for storing the read data during the transparent march elements S(1) to S(5)). Then, this value is again injected to the signature analyzer during the write operation following the read. This way, similarly to the march elements S(1) to S(4), the value v or v! is twice injected to the signature analyzer in each element of Pred. We also observe in figure 2.b, that the values read in the march elements of sequence Pred are the inverse of the values read in the march elements S(1) to S(5). Thus, during sequence Pred we inverse the read values before injecting them in the signature analyzer. This way, during fault-free operation, sequence Pred and march elements S(1) to S(5) inject the same values in the signature analyzer, resulting in equal sequences as required for signature prediction. Note finally that, in figure 2.b, at the end of march element S(5), the contents of the memory are the inverse of its initial content. Thus, sequence Rest restores the initial content. Proposition 3 shows that the transparent test algorithm of figure 2.b guaranties detection of all faults detected by MSS1.

S(0): ⇳(W0); S(1): ⇑(R0, R0, W1, W1);
S(2): ⇑(R1, R1, W0, W0); S(3): ⇓(R0, R0, W1, W1);
S(4): ⇓(R1, R1, W0, W0); S(5): ⇳(R0);

Figure 1: MSS1 test algorithm



(a) **(b)**

Figure 2: standard (a) and modified (b) transparent tests

Proposition 3: The transparent test of figure 2.b detects all static unlinked single-cell and two-cell FFMs and eliminates any masking due to the signature prediction principle.

Proof: MSS1 detects all static unlinked single-cell and two-cell FFMs. Then, from [40], march elements S(1) to S(5) in figure 2.a also detect these faults. From [40] this coverage does not depend on the memory contents. March elements S(1) to S(5) in figure 2.b are identical to those in figure 2.a, except that they start with inverse memory contents (v! versus v). Thus, they also detect all static unlinked single-cell and two-cell FFMs. On the other hand, the signature prediction will mask a fault if it produces errors in the same positions in Pred and in the march elements S(1) to S(5). A FFM can produce an error in a memory cell ci when the correct value of ci has a particular value vi. Then, as in figure 2.b the correct values that take a memory cell in the same positions of Pred and of march elements S(1) to S(5) are inverse to each other, a FFM cannot produce errors in the same positions of Pred and of march elements S(1) to S(5). As a consequence, the only way to have errors in the same positions of Pred and of march elements S(1) to S(5), is that cell ci is affected by two FFMs in which the sensitizing values of the victim cell (i.e. of ci) are inverse. However, in this case the two faults are linked and they are not part of the comprehensive set of all static unlinked functional fault models (FFMs) involving one memory cell (single-cell FFMs) and two memory cells (two-cell FFMs) [29], neither covered by MSS1.

QED

Corollary 1: Proposition 2 guaranties that in the case of the transparent test algorithm of figure 2a, no fault can be masked by the signature prediction scheme except the faults that are sensitized by read operations. This result is valid for both the conventional use of transparent test and its use in the context of ECC-based repair.

Corollary 2: Proposition 3 guaranties that in the case of the transparent test algorithm of figure 2b, no fault can be masked by the signature prediction scheme. This result is valid for the conventional use of transparent test. However, in the case of its use in the context of ECC-based repair we inject in the signature analyzer the syndromes of the detected errors rather that the errors themselves. Thus, fault masking becomes possible if different errors produced in the core test algorithm and the signature prediction sequence have identical syndromes.

Corollary 2 accredits that fault masking, due to the signature prediction scheme, is possible in the diagnosis of the ECC-based repair approach even for the algorithm 2.a. However, this masking is extremely rare due to the following reasons. First, in the proposed diagnosis scheme for ECC-based repair, if some word contains two faulty cells, then, if the two faulty cells are detected by the same read the double error will be detected by the ECC, leading to correct diagnosis decision: the memory contains words comprising two or more faulty cells. Furthermore, if the two faulty cells produce errors in different reads, then, as each of these reads will contain a single error, the ECC will produce error syndromes that correctly identify each of

the faulty cells. Thus, as these syndromes are cumulated (by XORing them) in the data field of a location of the diagnosis CAM, the word containing the two faulty cells will be identified by reading the diagnosis CAM. Therefore, if the memory contains some word(s) comprising two faulty cells the diagnosis decision will be correct. Then, miss-diagnosis can occur only if all the following conditions are satisfied:

a. *Some faulty memory words comprise more than two faulty cells and no memory words comprise two faulty cells*: because, on the one hand, only words comprising more than two faulty cells are not correctly diagnosed by the ECC and the diagnosis CAM, thus such faults should necessarily affect some memory word(s), and on the other hand, as shown earlier, if some other word contains two faulty cells correct diagnosis is also guaranteed.

b. *For each word comprising more than two faulty cells (which, due to point a, should necessarily exist if miss-diagnosis has to happen), no read detects exactly two faults*: because if this happens the double error would be detected by the ECC, signaling the presence of a word containing more than one faulty cell.

c. *For each word comprising more that two faulty cells, the case where a read detects one faulty cell and another read detects another faulty cell does not occur*: because if this happens the diagnosis CAM will identify a word containing at least two faulty cells.

d. *For each word comprising more than two faulty cells, there is at least one read of the core test algorithm detecting more than two faulty cells*. This is because as the core test algorithm covers all the considered faults each of them will be detected by some read, and since from c at most one read detects exactly one of them, then, there is at least one read of the core test algorithm detecting two or more faulty cells. Then, as from b no read detects exactly two of them, there is at least one read of the core test algorithm detecting more than two faulty cells.

e. *For each read of the core test algorithm detecting more that 2 faulty cells in a word comprising more than two faulty cells* (which read, due to point d, should necessarily exist if miss-diagnosis has to happen), *the corresponding read of the signature prediction sequence detects exactly one faulty cell or more than two faulty cells* (because if no faulty cell is detected masking due to the signature prediction could occur, and also because the detection of two faulty cells leads to correct diagnosis thanks to the ECC), *and the errors produced by these reads*, which errors are necessarily different to each other (because the faults are linked and the structure of the algorithm of figure 2.b prevents detecting the same cell in these two reads), *have identical syndromes* (because as these errors are different to each other and their syndromes are injected to the signature analyzer, the only possibility to have masking due to the signature prediction scheme is that these different errors give identical syndromes).

We observe that, some masking due to the signature prediction scheme is possible, but it can occur only if all the above complex conditions are satisfied. Thus, with the proposed approach, fault masking due to the signature prediction scheme is extremely rare. Due to time limitations the masking probability was not determined in the context of this thesis, but will be computed soon by means of analytical expressions and/or pseudo-simulations using the approach of detection profiles developed in chapter 3. However, in the present context, these detection profiles will be based on the detection operations presented in tables 1 and 2 of chapter 3, instead of the detection sequences used in the evaluation of the iterative diagnosis approach.

Comment 1: the above discussion concerns unlinked faults. Let us now consider the case of test algorithms covering also the linked faults. In this case, some reduction of the fault coverage is possible due to masking related to the signature prediction scheme. However, this masking is very improbable. Indeed, conditions a, b, c, and d, do not depend on the linked or unlinked property of the faults. Thus all of them should be satisfied whatever are the faults: unlinked; linked; or combinations of linked and unlinked. In addition for words affected only by unlinked faults, condition e should also be satisfied, while for all other words, the condition e' described below should also be satisfied.

Condition e'. *For each read operation of the core test algorithm detecting more that 2 faulty cells in a faulty word comprising more than two faulty cells* (which faulty word and read *operation* should

necessarily exist if miss-diagnosis has to happen, due to points a and d), *either all these faulty cells are affected by linked faults having opposite victim-cell sensitizing values and produce identical errors in the core test algorithm and in the signature prediction sequence, or different errors are produced in the read operation of the core test algorithm and in the corresponding read of the signature prediction sequence, and the ECC produces the same syndromes for these errors*.

Comment 2: Further to the masking analyzed in proposition 3, signature analysis introduces error aliasing. However, this is common to all signature analysis schemes, its value is very low, and can be reduced at will by increasing the size of the signature.

Comment 3: The transparent test algorithm considered above uses only one test data (all 0's vector) and its inverse (all 1's vector). However, it may be suitable to use more test data. For instance, if we desire testing couplings between consecutive cells of a word we can execute MSS1 with the all 0's and the all 1's vectors, as in figure 1, then, execute MSS1 a second time by using the 0101 …01 and the 1010 …10 vectors. The transparent test algorithm of figure 2.b can be extended to cover this case. To do this we can execute the algorithm of figure 2.b a first time. Then, we use a march element that reads each memory word, inverses all even positions of the read data and writes back the result in the same word. Then we start with the new memory contents and we execute the algorithm of figure 2.b a second time, except for sequence Rest in which we inverse only the odd positions of each memory word.

This approach can be trivially extended to algorithms using as test data any binary vector V_i and its inverse, as described next: we use a march element that reads each memory word, inverses all positions of the read data in which V_i is equal to 1 (i.e. we bit-wise XOR the read data with V_i) and we write back the result in the same word. Then we start with the new memory contents and we execute the transparent test algorithm in the conventional manner. We can repeat this as many times as the number of vectors that has to be used as test data. At the end we restore the initial contents by considering the vector V_j used as test data in the last write operation of the test algorithm, and adding a memory content restoring march element, which reads each memory word, inverses all positions of the read data in which V_j is equal to 1 and writes back the result in the same word.

ECC Consistency: To correctly test the memory, all bits in a word (data bits and ECC check bits) should undergo the transformations of the transparent test algorithm. However, these transformations will produce non-codewords inducing false error detections by the ECC. For instance, in the transparent test of figure 2.b, when the read value in the test algorithm is v!, injecting this value to the ECC decoder will produce false error detections, as inverting all bits of a codeword will give a non-codeword. To ensure ECC consistency, in the march elements S(1) and S(3) the read data are inverted before being injected to the ECC decoder. This is extended trivially in transparent tests derived from standard tests using as test data any subset of binary test vectors. In this case, at each instance of the transparent test algorithm, the read data will be inverted before being injected to the ECC decoder in the bit positions that have inverse values with respect to the initial memory content. That is, if vector V_j is the test data used in the latest write operation before the current read, then, the read data will be bit-wise XORed with V_j and the result will be injected in the ECC circuitry.

6.2.4 Transparent BIST for SRDF Test Algorithms

The previous developments consider the use of a diagnosis CAM. To eliminate this CAM and its associated cost we can the use SRDF test algorithms developed in chapter 2. The transparent BIST approach developed in the previous sections can be easily adapted to the case of SRDF test algorithms as described next.

First, these algorithms use multiple binary vectors as test data. Thus, the transparent test algorithm can be derived from an SRDF test algorithm as described in comment 3.

Second, SRDF algorithms guarantee that for any memory word comprising more than one faulty cell, there is always a read operation that detects at least two of them. Then we have two possible cases:

- This read operation contains a double error. In this case the ECC will detect this error and will signal that

there is a memory word containing at least two faulty cells.

- This read operation contains more than two errors. In this case the ECC may:
  - Not detect the existence of an error. Then, erroneous data are injected in the signature analyzer.
  - Detect the error and classify it as single. Then the ECC will miscorrect it, producing erroneous data that will be injected in the signature analyzer.
  - Detect the error and classify it as double. Then the ECC will signal that there is a memory word containing at least two faulty cells.

All cases produce either incorrect signature, which indicates the existence of a word comprising multiple faulty cells, or indicate by the ECC a word comprising two or more faulty cells. Thus, if a memory contains words comprising two or more faulty cells, the ECC and/or the signature analysis will discover their existence. In addition, if no memory word contains more than one faulty cell, only single errors will be produced by the read operations. These errors are correctly identified by the ECC as single errors and are also corrected. As the corrected data are injected to the signature analyzer, it will provide a correct signature. Thus, if no memory word contains more than one faulty cell, neither the ECC nor the signature will indicate the existence of words comprising multiple faulty cells.

Hence, for SRDF algorithms we can correctly diagnose the memory by using an approach in which:

- The ECC corrects the read data before injecting them in the signature analyzer.
- If the signature is incorrect and/or the ECC indicates the detection of more that one error in some read operation, the diagnosis outcome is that the memory contains words comprising multiple faulty cells. In all other cases the diagnosis outcome is that the memory does not contain words comprising multiple faulty cells.

6.3 CAM TEST AND REPAIR

Thanks to the adoption of the ECC-repair approach, the size of the run-time CAM is very small. The size of the diagnosis CAM is also small thanks to the block-based test and diagnosis strategy. For instance, as mentioned in section III.A, for a SRAM comprising 100K words of 32 bits we need a run-time CAM of only 48 locations, and if this memory is partitioned into 32 physical blocks we need a diagnosis CAM of only 186 locations. Thus, the probability of faults occurring in the CAMs will be much lower with respect to the probability of faults occurring in the SRAM. However, as we consider high defect densities, this probability may not be negligible. Therefore, both the run-time and the diagnosis CAMs should also be tested and repaired regularly.

Another issue is that there is no efficient protection against soft errors affecting the tag fields of a CAM. Under such a fault, incorrect misses can occur in the run-time CAM. Then, reads and writes will be performed over an SRAM word that comprises multiple faulty cells, leading to uncorrectable errors. In addition, soft errors cannot be detected by the test session in order to initiate the error recovery process.

To cope with these issues, we use the following CAM repair scheme [19][35][36][39]: each tag field of the runtime CAM and of the diagnosis CAM comprise a flag bit (repair-flag) used for repair purposes. This bit is set to 1 in the CAM locations discovered to be faulty by the test algorithm. The 1 value of this bit disables the match signal of the faulty location. Also, during memory repair, faulty addresses are stored only in tag fields in which the repair-flag bit value is 0. In addition, to cover the case where the repair-flag bit is faulty, two repair-flag bits can be used, guarantying that the match signal is disabled if any of them is 1 [19]. Another flag bit [36][39] is also used to indicate CAM locations occupied by faulty memory addresses (occupation-flag). Then, we use the following CAM test and repair strategy:

i. At the end of each test and repair session, the memory addresses stored in fault-free tag fields of the runtime CAM are transferred and stored in fault-free tag fields of the diagnosis CAM. This is feasible as the diagnosis CAM is much larger than the run-time CAM.

ii. At the beginning of the next test session, the memory addresses stored in the tag fields of good locations of the run-time CAM and of the diagnosis CAM are read and compared against each other. If the

comparison mismatches for some of the stored addresses, then: we check for each of the two mismatched addresses if the bank-address bits designate a memory bank that was used since the previous test session (i.e. the corresponding bit of BIR - defined in section 6.2.1 is equal to 1). If this is the case, for any of the two mismatched addresses, the error recovery process (described in section 6.1) is activated.

iii. In the next step, the diagnosis CAM is tested by means of CAM dedicated BIST (e.g. [52] [53]). In this BIST we use non-transparent test data, except for the repair-flag bits, for which we use transparent test data to preserve their values. In addition, the repair-flag bits in the locations discovered by the current test to be faulty are set to 1.

iv. The memory addresses stored in good locations of the run-time CAM (repair-flags = 0, occupation-flag =1), are transferred to fault-free locations of the diagnosis CAM (repair-flags = 0).

v. The run-time CAM is tested and repaired similarly to the diagnosis CAM. If some CAM location having its repair-flags equal to 0 is found to contain faults in the tag field or in its match mechanism, error recovery is activated (except if this process was already activated in step i). Error recovery is also activated if such a CAM location is found to contain more than two faulty cells in the data field.

vi. The addresses transferred in the previous step to the diagnosis CAM are transferred back to good locations of the run-time CAM.

vii. The transparent test and repair session of the SRAM (described in the previous sections) is activated.

Note that, the above strategy does not work for the approach based on the SRDF algorithms, as in this case no diagnosis CAM exists for realizing steps i, ii, and iv. Then, a small FIFO is implemented for supporting these steps. Each FIFO location consists into n+2 cells, where n is the number of the SRAM address bits. The extra two cells are used as repair-flag bits to signal faulty FIFO locations.

6.4 Conclusion

Ultimate CMOS and post-CMOS are expected to sharply accelerate circuit aging, resulting in very low MTBF (mean time between failures). Thus, frequent memory test sessions have to be activated during application execution. It requires using transparent memory BIST in order to preserve the contents of the memory. However, signature analysis cannot distinguish memory words comprising one faulty cell from those comprising multiple faulty cells, as required by ECC-based repair. In addition, ECC can incorrectly identify multiple errors as single errors or no errors. To cope with these issues, this chapter proposes a hybrid diagnosis scheme for transparent BIST, able to diagnose the existence of memory words containing multiple faulty cells. The two variants of this scheme work for ECC-based repair using diagnosis CAM as well as for ECC-based repair using the so-called SRDF test algorithms.

The second achievement of this chapter is the elaboration of the so-called block-based test and diagnosis scheme, which reduces both the test length and the size of the diagnosis CAM. A third achievement is the development of a signature prediction scheme for transparent memory test, which eliminates masking also for faults sensitized by read operations.

CHAPTER 7

CONCLUSION AND FURTHER DEVELOPMENTS

In modern SoCs embedded memories should be repaired to achieve acceptable yield. They should also be protected by ECC against field failures to achieve acceptable reliability. In technologies affected by high defect densities, conventional repair induces very high area and power penalties. To reduce them, we can take advantage of the ECC used for mitigating failures occurring in the field in order to also fix words comprising a single faulty cell, and use a word-repair scheme to fix all other faulty words (ECC-based memory repair). It was shown in previous works that ECC-based memory repair is the only repair approach able to repair memories affected by high defect densities at reasonable area cost. However, by analyzing the distribution of the detection instances of different faulty cells within the memory test algorithms, in this thesis we have shown that the low-cost benefits of ECC-based repair are lost due to diagnosis issues leading in complex diagnosis circuitry. We have also highlighted that, though ECC-based repair reduces dramatically power dissipation with respect to conventional (i.e. non-ECC-based) repair, this power is still significant in high defect densities. Thus, the aim of this thesis is to resolve these problems. Furthermore, since circuit aging is accelerated as we move towards the ultimate CMOS and post-CMOS technologies, leading to increasing field-failure rates, on-line test and repair is expected to become mandatory. Thus, another goal of this thesis is to develop an on-line test and diagnosis approach for preventive detection and repair of field faults may not be covered by the ECC.

In pursuing these goals, the developments accomplished in this thesis are the following.

In chapter 2, we introduced a new family of test algorithms that we coined as single-read double-fault detection (SRDF), which guarantee to detect in a single read at least two faulty cells of each word affected by two or more faulty cells. Then, we successfully addressed the complex theoretical challenges related with the development of algorithms satisfying the SRDF property. These test algorithms completely eliminate the diagnosis circuit, leading to drastic reduction of area cost. Thus, for high defect densities, ECC-based repair combined with SRDF test algorithms enables dramatic reduction of area and power cost with respect to conventional repair approaches. Beyond the dramatic reduction of power, which is of strategic importance due to the low-power constraints in modern technologies, in high defect densities, the extra area required for conventional repair but also for ECC-based repair (due to diagnosis issues), represents a high percentage of the memory area. As memories occupy the largest part of modern SoCs, usually more than 90%, this extra area represents a high percentage of the total chip area. Thus, its drastic reduction achieved by the SRDF algorithms is of high importance.

The drastic reduction of area penalty (achieved by the SRDF algorithms developed in chapter 2), allows low-cost repair of memories affected by high defect densities. However, this comes at the cost of significant increase of test length. To reduce test length, in chapter 3 we propose an architecture that uses

two separate CAMs: a large diagnosis-CAM sized to contain all faulty memory words and used during the test and diagnosis phase (which uses conventional test algorithms); and a small repair-CAM sized to contain only words comprising two or more faulty cells, and used at runtime for repairing bad memory words. The small repair-CAM used at runtime achieves the expected runtime power reduction of ECC-based repair, and the large diagnosis-CAM allows using conventional test algorithms, avoiding the test length increase related to SRDF test algorithms. However, as mentioned earlier, in high defect densities the large diagnosis-CAM induces inacceptable area penalty. Thus, this architecture is of practical interest for reducing runtime power dissipation with respect to conventional repair, but only for moderate defect densities. To allow trade-offs in terms of area and test time, in chapter 2 we also propose and formally prove an approach that reduces the size of the diagnosis-CAM, and compensates the missed CAM space by using an iterative diagnosis algorithm. This algorithm executes the test algorithm several times, and diagnoses at each iteration of the test algorithm a subset of the faulty memory words. The number of iterations depends on the size of the diagnosis-CAM and the number of faults and their distribution in the memory cells. To evaluate this approach and obtain statistically significant results, we need to perform large numbers of fault injections and simulations of the iterative diagnosis process. Due to the time-consuming process of memory fault simulation and of algorithmic simulation of CAM operation, these simulations cannot be accomplished at reasonable time. To cope with this issue, in chapter 3 we propose a pseudo-simulation approach, which determines the detection profile of each fault and its probability, and injects these profiles instead of the corresponding faults, avoiding performing fault simulation. This approach, together with a dedicated fast algorithm, which eliminates from the pseudo-simulation large parts of the iterative diagnosis process as well as the need for simulating the operation of the diagnosis CAM, enables very fast evaluation of the iterative diagnosis scheme. The outcome is that, the iterative diagnosis approach is of interest for low and intermediate defect densities: as the defect densities increase the number of iterations required for a given diagnosis-CAM size increase, resulting in both higher area and higher test length with respect to the SRDF test algorithms. Thus, the solutions developed in chapters 2 and 3 result in a comprehensive framework enabling trading area, power, and test time, to meet the application requirements and the defect densities of the target technology.

As mentioned earlier, though ECC-based repair reduces dramatically power dissipation with respect to conventional (i.e. non-ECC-based) repair, this power is still significant in high defect densities. As low-power is a stringent constraint in advanced technologies, further power reduction is a very important objective of this thesis. To achieve it, in chapter 4 we propose new, partitioned-based, repair architectures, which consider virtual partitions of the memory under repair (virtual memory blocks), and uses a virtual CAM dedicated to the repair of each virtual memory block. It results in a set associative memory, in which each set corresponds to one of these virtual CAMs. As a further improvement of this scheme, we add a CAM block (Overflow-CAM) to repair memory words, which are left unrepaired in virtual memory blocks that concentrate more faults than the majority of virtual memory blocks (due to the standard deviation of the statistical distribution of the faults, which is increased because of this partitioned). Then, for further power-dissipation reduction, we propose another architecture that replaces the Overflow-CAM by an Overflow Set-Associative Cache. The evaluation of these architectures is done by complex analytical yield-computation expressions, which result in intractable computation time. New yield-computation mathematics developed in chapter 5, results in fast yield computation for these expressions, and enabled the evaluation the new repair architectures. These evaluations show that, the developments proposed so far make possible the repair of memories affected by high defect densities at low area and power cost.

For the evaluation of the memory repair approaches proposed in chapters 1, 2, 3, and 4, we need to determine the size of the CAMs/Caches required for achieving a target yield for both the ECC-based repair and the non-ECC repair implemented by means of the proposed architectures. The analytical computation of the yield becomes increasingly complex for multiple reasons:

- In low defect densities, the faults affecting the repair CAM have negligible impact on the yield. But in high defect densities this impact is significant, requiring more complex yield computation.

- In high defect densities the size of the repair CAM becomes very high, increasing drastically the number of operations required to compute the yield.
- The introduction of sophisticated memory repair architectures, as the ones presented in section 4, requires very complex yield computation expressions.

To cope with these issues, in chapter 5 we derived several recursive relations enabling reducing by many orders of magnitude the number of operations required for computing the yield for the conventional, as well as for the ECC-repair architectures, developed in chapters 2 and 3. The situation is more complex for the partitioning-based repair architectures developed in chapter 4. Fortunately we discovered new yield-computation mathematics targeting these architectures, which lead to dramatic reduction of the computation complexity and enable fast yield computation even for the most complex of the proposed architectures, very large memories and very high defect densities.

Finally, the use of the proposed solutions in the context of application execution requires developing a transparent BIST approach that is compatible with the proposed ECC-based repair architectures. However, in transparent BIST, test data are verified by means of signature analysis instead of test data comparison. Signature analysis masks the information concerning the positions of the faulty cells, and does not allow identifying memories containing words affected by multiple faults. At the same time, ECC may misdiagnose memory words containing more than two faulty cells (i.e. it may identify them as containing one faulty cell). To cope with this issue, we developed a transparent BIST approach, which injects in the signature analyzer the error syndromes instead of the test data, and diagnoses the memory by considering at the same time the final signature, the responses of the ECC circuitry, and the content of the diagnosis CAM. We proved that this scheme diagnoses correctly all fault cases. Thus, the developed transparent BIST enables activating the test, diagnosis, and repair process in the context of the application execution, in order to cope with high rates of field-failure.

7.1 Goals' Accomplishment and Further Developments

In addition to the solutions proposed and evaluated in this thesis, in chapters 4 and 6 we also propose several promising solutions, which were not yet evaluated due to time limitations. These solutions will represent in our future developments promising extensions of the presented work. They comprise:

- An Overflow Cache conditional-selection approach, which allows disabling most of the time the Overflow Cache (CACHE2). This disabling, results in further reduction of power dissipation, whose source will mainly consist in the power dissipations of CACHE1, that we can reduce almost at will by increasing the number of sets and reducing their size (the number of ways).
- A block based diagnosis approach, which reduces drastically the size of the diagnosis-CAM, without increasing test length.

In combination with the other developments accomplished in the present work, they should enable memory repair for high-defect densities at very low area, power, and test time penalties. Thus we are close to the achievement of the goals announced in chapter 1, consisting in providing a repair approach that can be used for:

- Mitigating the impact of very high rates of fabrication and field faults on: fabrication yield, reliability, and product life duration;
- Reducing aggressively the operating voltage in order to achieve drastic power reduction;

As stated earlier, a last goal of this thesis was to provide an ECC-based repair approach that can be used in the context of application execution. This was accomplished in chapter 6, by developing a transparent memory BIST approach compatible with ECC-repair. This approach induces some aliasing, which was shown to be very improbable, but its exact computation was not performed in the context of the present thesis because of time limitations. This evaluation is scheduled for the immediate future.

Last but not least, disposing an approach coping with very high failure rates in memories is certainly of high interest. However producing systems in highly defective technologies requires mitigating high

failure rates in all parts of the system (including memories, logic blocks, processors, routers, interconnections, …). The accomplishment of this goal is the aim of the *Cells* project, which addresses these issues in all levels of the system. Thus, the integration in the *Cells* framework of the architectures developed in this thesis, and their validation within this framework at block and system level, is also a major goal for our future developments, which will be accomplished by integrating these architectures together with the other *Cells* components in a scalable simulation infrastructure using the state of the art Structural Simulation Toolkit (SST).

List of Publications

P. Papavramidou, M. Nicolaidis, “Test Algorithms for ECC-based Memory Repair in Nanotechnologies”, IEEE VLSI Test Symposium (VTS), April 2012.

P. Papavramidou, M. Nicolaidis, “An Iterative Diagnosis Approach for ECC-based Memory Repair”, IEEE VLSI Test Symposium (VTS), April 2013.

P. Papavramidou, M. Nicolaidis, “Reducing Power Dissipation in Memory Repair for High Defect Densities”, IEEE European Test Symposiums (ETS), Mai 2013.

M. Nicolaidis P. Papavramidou, “Transparent BIST for ECC-based Memory Repair”, IEEE International On-Line Testing Symposium (IOLTS), July 2013.

Bibliography

[1] Zorian Y., “Embedded Memory Test & Repair: Infrastructure IP for SOC Yield“, 2002 IEEE International Test Conference.

[2] Sawada K., Sakurai T., Uchino Y., Yamada K., “Built-In self repair circuit for High Density ASMIC”, IEEE 1999 Custom Integrated Circuits Conference.

[3] Tanabe A. et al “ A 30-ns 64-Mb DRAM with Built-in Self-test and Self-Repair Function”, IEEE Journal Solid State Circuits, pp. 1525-1533, Vol 27, No 11, Nov. 1992.

[4] Benso A. et al “A Family of Self-Repair SRAM Cores”, 2000 IEEE International Test Conference. 2000 In Proc. IEEE International On-Line Testing Workshop, July 3-5, 2000.

[5] Kim I., Zorian Y., Komoriya G., Pham H., Higgins F. P., Newandowski J.L. "Built-In self repair for embedded high-density SRAM" Proc. Int. Test Conference, 1998, pp1112-1119

[6] V. Schober, S. Paul, O. Picot, “Memory Built-In Self-Repair using redundant words”, 2001 IEEE Intl Test Conference.

[7] J.-F. Li, J.-C. Yeh, R.-F. Huang, and C.-W. Wu,”A built-in self-repair design for RAMs with 2-D redundancies,” IEEE Trans. Very Large Scale Integration Systems, vol.13, no.6, pp. 742-745, June, 2005.

[8] S.-K. Lu, Y.-C. Tsai, C.-H. Hsu, K.-H. Wang, and C.-W. Wu, “Efficient built-in redundancy analysis for embedded memories with 2-D redundancy,” IEEE Trans. on VLSI Systems, vol. 14, no. 1, pp. 34–42, Jan. 2006.

[9] C.-D. Huang, J.-F. Li, and T.-W. Tseng, ”ProTaR: an infrastructure IP for repairing RAMs in SOCs,”, IEEE Trans. Very Large Scale Integration Systems, vol.15, no.10, pp. 1135-1143, Oct. 2007.

[10] Nicolaidis, M., Achouri, N., & Boutobza, S., “Optimal reconfiguration functions for column or data-bit built-in self-repair”. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2003* (pp. 590-595). IEEE.

[11] Nicolaidis, “Design for Soft-Error Robustness To Rescue Deep Submicron Scaling”, Proceedings Intl Test Conference 1998

[12] R.C. Baumann, "Soft Errors in Advanced Computer Systems," IEEE Design and Test of Computers, vol. 22, no. 3, pp. 258-266, May/June 2005.

[13] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, T. Toba, "Scaling Effects on Neutron-Induced Soft Error in SRAMs Down to 22nm Process".3rd Workshop on Dependable and Secure Nanocomputing, June 2009, Lisbon, Portugal.

[14] W. Wang, V. Balakrishnan, B. Yang, Y. Cao, “Statistical prediction of NBTI-induced circuit aging”, International Conference on Solid-State and Integrated-Circuit Technology, (ICSICT), October 23,

2008, pp. 416-419.
[15] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SECDED codes", IBM J. Res. Develop., vol. 14, pp. 395-401, July 1970.
[16] K. Gray, "Adding Error-Correcting Circuitry to ASIC Memory", IEEE Spectrum, pp. 55-60, Apr. 2000.
[17] S. Ghosh, S. Basu, N.A. Touba, "Selecting Error Correcting Codes to Minimize Power in Memory Checker Circuits", J. Low Power Electronics 1, pp.63-72 (2005).
[18] M. Nicolaidis, "Soft Errors in Modern Electronic Systems", Springer, Frontiers in Electronic Testing, Volume 41, 2011. ISBN 978-1-4419-6992-7
[19] M. Nicolaidis, N. Achouri, L. Anghel, "A Diversified Memory Built In Self Repair Approach for Nanotechnologies", IEEE VLSI Test Symposium/Best Paper Award, April-May 2004.
[20] Horiguchi M., Itoh K., "Nanoscale Memory Repair", Springer, Series: Integrated Circuits and Systems, 1st Edition., 2011.
[21] Itoh K., "Adaptive Circuits for the 0.5-V Nanoscale CMOS", Keynote ISSCC 2009.
[22] M. Nicolaidis, "Soft Errors in Modern Electronic Systems", Springer, Frontiers in Electronic Testing, Volume 41, 2011. ISBN 978-1-4419-6992-7; e-ISBN 978-1-4419-6993-4
[23] M. Nicolaidis, "Designing Single-Chip Massively Parallel Tera-Device Processors: Towards the Terminator Chip", Keynote Plenary Session, 2011 IEEE VLSI Test Symposium (VTS'11, Mai 1-5, Dana Point, California.
[24] M. Nicolaidis, "Designing Robust Single-Chip Massively-Parallel Tera-Device Processors", Opening Session Keynote, 4th Design for Reliability Workshop (DFR) - HiPEAC - Paris, France, January 23, 2012 - Paris, France, January 23, 2012
[25] M. Nicolaidis, "Biologically Inspired Robust Tera-Device Processors", IEEE Design & Test of Computers, Volume 29, No 5, September/October 2012.
[26] Nicolaidis, M. (2007, October). Graal: a new fault tolerant design paradigm for mitigating the flaws of deep nanometric technologies. IEEE International Test Conference, *2007. ITC 2007.* (pp. 1-10).
[27] G. Bizot, D. Avresky, F. Chaix, N.E. Zergainoh, M. Nicolaidis, "Self-Recovering Parallel Applications in Multi-Core Systems", 10th IEEE International Symposium on Network Computing and Applications (IEEE NCA11), 25 - 27 August 2011 Cambridge, MA USA
[28] S. Hamdioui, A.J. van de Goor, M. Rodgers, "March SS: A Test for All Static Simple RAM Faults", Proc. of the 2002 IEEE Intl Workshop on Memory Technology, Design and Testing
[29] A.J. van de Goor and Z. Al-Ars, "Functional Fault Models: A Formal Notation and Taxonomy", In Proc. of IEEE VLSI Test Symposium, pp. 281-289, 2000.
[30] R. Dekker et al, "Fault Modelling and test algorithm development for Static Random Access Memories", in Proc. IEEE International Test Conference pp. 343-352, 1998
[31] G. Harutunyan V.A. Vardanian Y. Zorian, "Minimal March Tests for Unlinked Static Faults in Random Access Memories", Proc. of the 23rd IEEE VLSI Test Symposium, May 1-5 2005.
[32] http://www.tamps.cinvestav.mx/~jtj
[33] J. Torres-Jimenez, E. Rodriguez-Tello, "Simulated Annealing for Constructing Binary Covering Arrays of Variable Strength", IEEE World Congress on Computational Intelligence, July 18-23 2010, Barcelona, Spain
[34] http://quid.hpl.hp.com:9081/cacti/
[35] P. Papavramidou, M. Nicolaidis, "Test Algorithms for ECC-based Memory Repair in Nanotechnologies", IEEE VLSI Test Symposium, April 2012.
[36] P. Papavramidou, M. Nicolaidis, "Reducing Power Dissipation in Memory Repair for High Defect Densities", IEEE European Test Symposiums (ETS), Mai 2013.
[37] Z. Al-Ars and A. van de Goor, "Static and dynamic behavior of memory cell array spot defects in embedded drams," IEEE Trans. on Comp., vol. 52, no. 3, pp. 293–309, 2003.
[38] M. Zhang, K. Asanovic, "Highly-Associative Caches for Low-Power Processors", Kool Chips

Workshop, 33rd International Symposium on Microarchitecture, Monterey, CA, December 2000.
[39] P. Papavramidou, M. Nicolaidis, "An Iterative Diagnosis Approach for ECC-based Memory Repair", 2013 IEEE VTS.
[40] M. Nicolaidis, "Theory of transparent BIST for RAMs", IEEE Transactions on Computers, 45.10 (1996): 1141-1156.
[41] M. Nicolaidis, "Transparent BIST for RAMs". In : *Proc. 1992 International-Test-Conference,* p. 598-607.
[42] I. Voyiatzis, "An accumulator-based compaction scheme for online BIST of RAMs", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 16.9 (2008): 1248-1251.
[43] S. Hellebrand, H. J. Wunderlich, A. A. Ivaniuk, Y. V. Klimets, V. N. Yarmolik, "Efficient online and offline testing of embedded drams" , *IEEE Transactions on Computers, 51*.7 (2002), 801-809.
[44] K. Thaller, A. Steininger. "A transparent online memory test for simultaneous detection of functional faults and soft errors in memories", *IEEE Tr. on Reliability,* 52.4 (2003): 413-422. [?????]
[45] D. C. Huang, W. B. Jone. "A parallel transparent BIST method for embedded memory arrays by tolerating redundant operations", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21.5 (2002): 617-628.
[46] Li, Jin-Fu. "Transparent-test methodologies for random access memories without/with ECC", *IEEE Tr. on CAD of Integrated Circuits and Systems,* 26.10 (2007): 1888-1893.
[47] P. Camurati, P. Prinetto, M. S. Reorda, S. Barbagallo, A. Burri, D. Medina, "Industrial BIST of embedded RAMs", *IEEE Design & Test of Computers*, *12*.3 (2010), 86-95.
[48] C. Weaver, A. Todd, "A fault tolerant approach to microprocessor design." IEEE *International Conference on Dependable Systems and Networks (DSN)*, 2001.
[49] A. Steininger, C. Temple, "Economic online self-test in the time-triggered architecture." *IEEE Design & Test of Computers,* 16.3 (1999): 81-89.
[50] M. Nicolaidis, L. Anghel, N. E. Zergainoh, D. Avresky, "Designing Single-Chip Massively Parallel Processors Affected by Extreme Failure Rates", Proc. Design Automation and Test in Europe Conference, pp. 679-682, March 12 – 16, 2012, Dresden.
[51] C. Rusu, C. Grecu, L. Anghel, "Improving the scalability of checkpoint recovery for networks-on-chip", IEEE Intl Symp. on Circuits and Systems (ISCAS), 2008, Seattle, Washington, USA.
[52] K. J. Lin and C. W. Wu, "Testing Content-Addressable Memories Using Functional Models and March-Like Algorithm," *IEEE Trans. Computer-Aided Des. of Integrated Circuits and Systems*, vol. 19, no. 5, May 2000. pp. 577-588.
[53] Y.S.Kang,J.C.Le,and S.Kang,"Paralel BIST architecture for CAMs,"*Electronics Letters,* vol. 33, no. 1, pp. 30–31, Jan. 1997.
[54] Kim H. C. et al, "A BISR (Buil-In Self-Repair) circuit for embedded memory with multiple redundancies", 1999 IEEE International Conference on VLSI and CAD, Oct. 26-27, 1999.
[55] M. Nicolaidis, N. Achouri, S. Boutobza, "Dynamic Data-Bit Memory Built-In Self-Repair", Intl Conference on Computer Aided Design (ICCAD), November 2003, San Jose, CA, USA
[56] S. Thoziyoor, J. Ahn, M. Monchiero, J. Brockman, and N. Jouppi, "A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies", 35th International Symposium on Computer Architecture (ISCA), June 2008, Beijing, China
[57] S. Li, *et al.*, "CACTI-P: Architecture-Level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques", IEEE/ACM International Conference on Computer-Aided Design (ICCAD), November 2011, San Jose, US
[58] E. Rodriguez-Tello, J. Torres-Jimenez, "Memetic Algorithms for Constructing Binary Covering Arrays of Strength Three" Lecture Notes in Computer Science (LNCS) 2010, Volume 5975, pp 86-97

Abstract - Nanometric scaling increases the sensitivity of integrated circuits to defects and perturbations. Thus, each new generation of manufacturing process is accompanied by a rapid degradation of manufacturing yield and reliability. Embedded memories occupy the largest part of the area of SoCs and comprise the vast majority of transistors. In addition, for increasing the integration density, they are designed very tightly to the design and electrical rules. Hence, embedded memories concentrate the majority of the manufacturing defects affecting a SoC, and are also more sensitive to perturbations. Thus, they are the parts of the SoC the most affected by the deterioration of manufacturing yield and reliability. This thesis develops repair architectures optimally combining test algorithms, BIST architectures, and error correcting codes, in order to propose effective solutions for improving the manufacturing yield and reliability of embedded memories affected by high defect densities.

Keywords – memory testing, memory repair, reliability, yield, high defect densities

---

Resumé - La miniaturisation technologique augmente la sensibilité des circuits intégrés aux défauts et nous observons à chaque nouvelle génération technologique une dégradation rapide du rendement de fabrication et de la fiabilité. Les mémoires occupent la plus grande partie de la surface des SoCs et contiennent la vaste majorité des transistors. De plus, pour augmenter leur densité elles sont conçues de façon très serrée. Elles concentrent ainsi la plus grande partie des défauts de fabrication et représentent aussi les parties les plus sensibles face aux perturbations. Elles sont par conséquent les parties des SoCs les plus affectées par la dégradation du rendement de fabrication et de la fiabilité. L'objectif de cette thèse est de proposer des architectures combinant de façon optimale : algorithmes de test, architectures BIST, et codes correcteurs d'erreurs afin de proposer des solutions efficaces pour l'amélioration du rendement de fabrication et de la fiabilité des mémoires embarquées.

Mots clés – test des mémoires, réparation des mémoires, fiabilité, rendement, hautes densités de défauts

---

Laboratoire TIMA, 46, avenue Félix Viallet, 38031 Grenoble Cedex, France

ISBN 978-2-11-129195-9